
# NAVAL POSTGRADUATE SCHOOL

**MONTEREY, CALIFORNIA**

# THESIS

**REFINEMENT COMPOSITION USING DOUBLY LABELED TRANSITION GRAPHS**

by

Thor Martinsen

September 2007

Thesis Co-Advisors: George Dinolt
Harold Fredricksen

**Approved for public release; distribution is unlimited.**

THIS PAGE INTENTIONALLY LEFT BLANK

| REPORT DOCUMENTATION PAGE | | | Form Approved OMB No. 0704-0188 |
|---|---|---|---|
| Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503. | | | |
| **1. AGENCY USE ONLY** *(Leave blank)* | **2. REPORT DATE** September 2007 | **3. REPORT TYPE AND DATES COVERED** Master's Thesis | |
| **4. TITLE AND SUBTITLE** Refinement Composition Using Doubly Labeled Transition Graphs | | **5. FUNDING NUMBERS** | |
| **6. AUTHOR(S)** Thor Martinsen | | | |
| **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Naval Postgraduate School Monterey, CA 93943-5000 | | **8. PERFORMING ORGANIZATION REPORT NUMBER** | |
| **9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)** N/A | | **10. SPONSORING/MONITORING AGENCY REPORT NUMBER** | |
| **11. SUPPLEMENTARY NOTES** The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. | | | |
| **12a. DISTRIBUTION / AVAILABILITY STATEMENT** Approved for public release; distribution is unlimited | | **12b. DISTRIBUTION CODE** | |

**13. ABSTRACT (maximum 200 words)**

Process Algebra forms a cornerstone in the formal methods area of Computer Science. Among the more widely used approaches is Milner's Communication and Concurrency Systems (CCS). Recently CCS has been extended by Schmidt and Bibighaus through the introduction of Doubly Labeled Transition Systems. This framework has enhanced the model's ability to capture security and availability properties. In this thesis we reformulate, simplify, and extend Bibighaus' work using a graph theoretic framework. The intent is that this abstract mathematical view will make the results more accessible and stimulate additional research. Existing definitions and theorems are redefined and proved using Labeled and Doubly Labeled Transition Graphs (LTG and DLTG). CCS simulation concepts are recast as graph morphisms and the notion of abstraction and refinement are explained through the use of graphs. Bibighaus' work is extended by showing how to carry out non-atomic DLTG refinement, and by developing a form of graph composition involving graph refinements that share a common abstract graph. This type of composition is proven to always be possible with DLTG refinements, and we demonstrate that the composite graph is both a refinement of the abstract graph, and an abstract graph for the refinements from which it was made.

| **14. SUBJECT TERMS** Security, Formal Methods, Refinement, High Assurance | | | **15. NUMBER OF PAGES** 77 |
|---|---|---|---|
| | | | **16. PRICE CODE** |
| **17. SECURITY CLASSIFICATION OF REPORT** Unclassified | **18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified | **19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified | **20. LIMITATION OF ABSTRACT** UU |

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

**Approved for public release; distribution is unlimited.**

**REFINEMENT COMPOSITION USING DOUBLY LABELED TRANSITION GRAPHS**

Thor Martinsen
Lieutenant Commander, United States Navy
B.S., Thomas Edison State College, 1996

Submitted in partial fulfillment of the
requirements for the degrees of

**MASTER OF SCIENCE IN COMPUTER SCIENCE**

and

**MASTER OF SCIENCE IN APPLIED MATHEMATICS**

from the

**NAVAL POSTGRADUATE SCHOOL**
**September, 2007**

Author: Thor Martinsen

Approved by: George Dinolt
Thesis Co-Advisor

Harold Fredricksen
Thesis Co-Advisor

Peter J. Denning
Chairman, Department of Computer Science

Clyde Scandrett
Chairman, Department of Applied Mathematics

THIS PAGE INTENTIONALLY LEFT BLANK

# ABSTRACT

Process Algebra forms a cornerstone in the formal methods area of Computer Science. Among the more widely used approaches is Milner's Communication and Concurrency Systems (CCS). Recently CCS has been extended by Schmidt and Bibighaus through the introduction of Doubly Labeled Transition Systems. This framework has enhanced the model's ability to capture security and availability properties. In this thesis we reformulate, simplify, and extend Bibighaus' work using a graph theoretic framework. The intent is that this abstract mathematical view will make the results more accessible and stimulate additional research. Existing definitions and theorems are redefined and proved using Labeled and Doubly Labeled Transition Graphs (LTG and DLTG). CCS simulation concepts are recast as graph morphisms and the notion of abstraction and refinement are explained through the use of graphs. Bibighaus' work is extended by showing how to carry out non-atomic DLTG refinement, and by developing a form of graph composition involving graph refinements that share a common abstract graph. This type of composition is proven to always be possible with DLTG refinements, and we demonstrate that the composite graph is both a refinement of the abstract graph, and an abstract graph for the refinements from which it was made.

THIS PAGE INTENTIONALLY LEFT BLANK

# TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF SYMBOLS

| | | |
|---|---|---|
| $V$ | Vertex set | 4 |
| A | Set of actions | 4 |
| $E$ | Set of labeled edges | 4 |
| $v_0$ | Start vertex | 4 |
| $\langle V_G, \mathrm{A}_G, E_G, v_0 \rangle$ | Labeled Transition Graph $G$ | 4 |
| $p \xrightarrow{\alpha} q$ | Transition from vertex $p$ to vertex $q$ on action $\alpha$ | 4 |
| $\alpha(p) = q$ | Transition from vertex $p$ to vertex $q$ on action $\alpha$ | 4 |
| $P_v$ | Directed path in a graph starting at vertex $v$. | 5 |
| $\eta(e)$ | Function that returns a head vertex of an edge $e$ | 5 |
| $\tau(e)$ | Function that returns a tail vertex of an edge $e$ | 5 |
| $G\|_{P_v}$ | The graph $G$ restricted to $p_v$ | 5 |
| $P(G)$ | The set of directed paths of graph G | 5 |
| $\langle a_n \rangle$ | Sequence of actions in a trace | 5 |
| $t_v$ | Trace with respect to vertex $v$ | 5 |
| $T(G)$ | Trace set of graph $G$ | 6 |
| $G_1 \overset{trace}{\equiv} G_2$ | Graphs $G_1$ and $G_2$ are trace equivalence | 6 |
| $R$ | Simulation, Bisimulation, or MCB relation | 7 |
| $G_B \lesssim_R G_A$ | Simulation of $G_A$ by $G_B$ on relation R | 7 |
| $G_A \approx_R G_B$ | Bisimulation relation $R$ between $G_A$ and $G_B$ | 8 |

| | | |
|---|---|---|
| $R_{\rightarrow} \subseteq (E_A, E_B)$ | Relation $R$ ensures $E_A \subseteq E_B$ | 8 |
| $\tilde{\tilde{G}}$ | Set of bisimulation equivalent graphs | 11 |
| \|p\| | The length of a directed path $p$ | 14 |
| $V \dot{\prec} v$ | Vertex refinement of $v$ by a set of Vertices $V$ | 17 |
| $G_C \vec{\prec} e$ | Edge refinement of e by the graph $G_C$ | 18 |
| □ | Must label | 23 |
| ◇ | May label | 23 |
| $\langle V_G, \mathrm{A}_G, E^{\diamond}, E^{[]}, v_0 \rangle$ | Doubly Labeled Transition Graph $G$ | 23 |
| $E^{\diamond}$ | A set of doubly labeled "May" edges | 23 |
| $E^{[]}$ | A set of doubly labeled "Must" edges | 23 |
| $G^{[]}$ | □ induced subgraph of the DLTG $G$ | 24 |
| $G_C^{[]} \mid_{R^{-1}(E^{[]}_{G_A})}$ | $G_C^{[]}$ restricted with respect the image of $G_A$ under $R^{-1}$. | 24 |
| $G_A \mid\!\approx_R G_C$ | Modal Constrained Bisimulation under $R$ | 24 |

# LIST OF DEFINITIONS

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF LEMMAS

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF THEOREMS

THIS PAGE INTENTIONALLY LEFT BLANK

# ACKNOWLEDGMENTS

First and foremost, I would like to thank my wife, Dione, and son, Corey. This would neither have been possible, nor worth while if not for their love and support.

Secondly, I would be remiss in my duties were I to neglect recognizing Dr. George Dinolt and Dr. Hal Fredricksen for their mentorship and encouragement. I could not have hoped for better advisors. It is an honor to have studied under them.

Finally, I would like to thank the United States Navy for allowing me to attend this great institution. John Adams once said: "I must study politics and war that my sons may have liberty to study mathematics and philosophy." It is my sincere hope that the time and latitude I have been afforded to pursue the latter, will improve my ability to carry out my duties related to the former.

THIS PAGE INTENTIONALLY LEFT BLANK

# I. INTRODUCTION

## A. PROBLEM STATEMENT

In order to develop high assurance software under the Common Criteria Certification requirements [1] it is necessary to adopt a layered development approach. Formal methods are frequently used in this process because they provide a means of developing and proving that an abstract model of a system is secure. This model is then refined until a concrete implementation is achieved. At each layer of the development process, one is able to assert the security of the refinement by mapping it back to the abstract model. Unfortunately, despite this systematic development approach, it still is possible that the concrete system contains security flaws such as covert channels.

Recently Bibighaus [2] and others have introduced Doubly Labeled Transition Systems, and have demonstrated how this model is able to guarantee that a larger set of security and availability properties are retained throughout the refinement process.

The purpose of this thesis is twofold: First, we restate and simplify Bibighaus' research using a graph theoretic framework that is more accessible and widely understood than the Prototype Verification System (PVS) environment used throughout his dissertation. Secondly, we address the following question: Given an abstract specification and two different refinements of that specification, is it possible to compose the two refinements in such a manner as to create a new refinement that captures the properties of both refinements while still satisfying the abstract specification?

## B. MOTIVATION

Bibighaus' work on Doubly Labeled Transition System has demonstrated the model's potential as a useful tool in the secure software development process. However, much work still remains. One unanswered question is how to go about composing two refinements of the same abstract specification. This question is relevant because it is

conceivable that during the development process, two different refinements are created that both capture desirable aspects of a system and hence need to be brought together to form a more complete refinement.

By presenting Bibighaus' work using a graph theoretic framework it is hoped that we can make his results accessible to a wider audience and stimulate additional research in this important area of software engineering.

## C. METHODOLOGY

This research is conducted using graph theory. The existing Doubly Labeled Transition System framework utilized by Bibighaus is redefined and explained in graph theoretic terms.

## D. CONTRIBUTION

This thesis makes the following contributions.

1. We explain Labeled and Doubly Labeled Transition Systems using a graph theoretic framework. Existing definitions and theorems from Bibihaus' work are redefined and proved using Labeled and Doubly Labeled Transition Graphs.
2. We prove that edge refinement of a "Must" edge in a DLTG, requires that every edge of the graph that refined that edge also is a "Must" edge.
3. We develop a method of combining Labeled and Doubly Labeled Transition Graph refinements that share a subgraph and a common abstract graph. Moreover, we show that this process is always possible to do with DLTGs.
4. We prove that the join of two DLTG refinements that share a common abstract graph is a refinement of the abstract graph, and that the new graph also serves as an abstract graph for the two refinements used to create it.

# II. PRELIMINARIES

In this chapter we briefly discuss Process Algebras, and then introduce Labeled Transition Graphs. This is the fundamental mathematical structure that will be used throughout this thesis. Finally, we discuss equivalence classes of Labeled Transition Graphs, and show how this concept could be leveraged when trying to prove equivalence between different graphs.

## A. PROCESS ALGEBRAS

Process Algebras have seen wide use in the field of computer science. The reason for this is that they provide a mathematical framework necessary to formally reason about computer behavior. Process Algebras allow us to model computer interactions, communications, and synchronizations through the use of algebraic rules that can be manipulated and analyzed. Among the most popular Process Algebras are Communicating Sequential Processes (CSP) developed by C.A.R. Hoare [3] and Robin Milner's Communication and Concurrency Systems (CCS) [4]. Because of their wide-spread use, both of these frameworks have been thoroughly analyzed and proven to be Turing complete. CSP is a Process Algebra which uses denotational semantics. Here a process is represented as a set of mathematical objects. CCS on the other hand, uses operational semantics. In this system, a process is expressed as a set of actions that can occur. CCS is frequently represented using Labeled Transition Systems. A Labeled Transition System consists of a set of system states and a set of labeled transitions between these states. Recently CCS has been extended by Schmidt [5] and Bibighaus [2] to support Doubly Labeled Transition Systems. This framework has enhanced the model's ability to capture safety and liveness properties in computer systems.

In this thesis we focus on the operational semantic approach of Bibighaus. His work was partially built upon that of Milner [4] and made extensive use of Labeled Transition Systems. His results were formulated and proven using the Prototype Verification System (PVS) theorem proving software [6]. Our goal is to present and extend his work using a graph theoretic approach.

## B. LABELED TRANSITION GRAPHS (LTG)

In this chapter we introduce Labeled Transition Graphs. These graphs are similar in many respects to Labeled Transition System. [2] However, all attributes of Labeled Transition Systems are now defined using graph theory. We can use a Labeled Transition Graph to model the behavior of a computer system. We will use a notation that makes this connection direct.

**Definition 2.1: (Labeled Transition Graph)** Let *G* be a directed graph defined by the quadruple $G = \langle V, \mathrm{A}, E, v_0 \rangle$ where

| | |
|---|---|
| $V$ | is a set of vertices called states. |
| A | is a set of labels called actions. |
| $E$ | is the set of labeled edges such that $E \subseteq V \times A \times V$ . |
| $v_0$ | is a distinguished start vertex such that $v_0 \in V$ . |

Then, given an LTG *G*, where $p, q \in V_G$ , $\alpha \in \mathrm{A}_G$ , and $(p, a, q) \in E_G$ we use the notation:

$$p \xrightarrow{\alpha} q \text{ or } \alpha(p) = q$$

We interpret this as follows: whenever vertex *p* in *G* is acted upon by action $\alpha$ , the system will transition to state *q* via the labeled edge (p, a, q).

As an example, consider a pushbutton flashlight. This is, in fact, a simple electrical machine with two states. The flashlight can either be on or off, and the action required to affect a change in state is a simple push action. We can model this machine using a Labeled Transition Graph with start vertex $v_{Off}$ as:

$$Flashlight = \left\langle V_{Flashlight} = \{v_{Off}, v_{On}\}, A_{Flashlight} = \{Push\}, E_{Flashlight} = \{v_{Off} \xrightarrow{Push} v_{On}, v_{On} \xrightarrow{Push} v_{Off}\}, v_{Off} \right\rangle$$

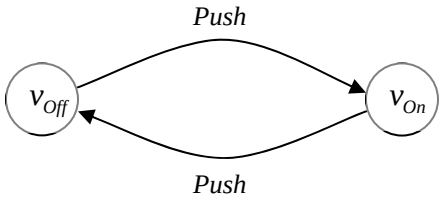



**Figure 1. LTG Model Of Flashlight Behavior**

Labeled Transition Graphs are intuitive, and can be very useful, particularly when attempting to model more complex systems.

**Definition 2.2: (A directed path in an LTG)** Let $G$ be an LTG, $v$ be a vertex in G, and $\eta$ and $\tau$ be functions that return the respective head or tail of a given edge. Then a directed path $p_v = \langle e_0, e_1, e_2, ..., e_n \rangle \ni e_i \in E_G$ is a sequence of labeled edges such that: $\eta(e_0) = v \wedge \forall i \in [0, n-1] \ \tau(e_1) = \eta(e_i + 1)$

Note that in Labeled Transition Graphs we allow for both repeated edges and vertices in directed paths. The directed path $p_v$ is a subgraph of $G$ and we denote the graph $G$ restricted to $p_v$ as $G|_{P_v}$.

**Definition 2.3: (The set of directed paths in an LTG)**

Let $G$ be an LTG with start vertex $v_0$. Then $P(G)$ is the set of all directed paths of G starting at $v_0$.

## 1. Traces

In Process Algebras we often would like to compare systems. One way to do this is by comparing the sequence of actions that different systems are capable of executing. These patterns of allowable actions are referred to as traces.

**Definition 2.4: (A trace in an LTG)** Given an LTG $G$ and a vertex $v$ in $G$, $t_v$ is a trace in G with respect to $v$ such that:

If $t_v = \langle a_i \rangle \ i = 0, 1, 2, ..., n-1, \quad \forall i, a_i \in A_G$ is a sequence of actions, then $\exists \ p_v = \langle e_0, e_1, e_2, ... e_n \rangle \in G \ni \langle a\langle e_0 \rangle, a\langle e_1 \rangle, a\langle e_2 \rangle, ..., a\langle e_n \rangle \rangle = t_v$

We use the notation $T(p_v) = t_v$ to represent the trace associated with the path $p_v$.

Depending on G, there may be many different traces starting at a given vertex in G. In fact, since our definition of a directed path allows for vertex and edge repetitions, there

could be infinitely many such traces if G contains a cycle. Using Definition 2.3 we construct the set of traces that a Labeled Transition Graph will "accept".

**Definition 2.5: (A trace set of an LTG)** Given an LTG *G* let $T(G)$ represent the set of all traces in *G* with respect to the start vertex $v_0$.

$T(G) \subseteq A_G^*$, where $A_G^*$ represents the Kleene closure of the set of actions for LTG *G*. Since a trivial walk, i.e. a walk of length 0, exists in any graph [7], we conclude that the empty sequence will be a part of any system modeled by an LTG.

One type of relationship we can establish between graphs is trace equivalence. Labeled Transition Graphs are trace equivalent if they have the same trace sets.

**Definition 2.6: (Trace equivalence between LTGs)** Given two labeled transition graphs $G_1, G_2$

$$G_1 \overset{trace}{\equiv} G_2 \Leftrightarrow T(G_1) = T(G_2)$$

Informally we say that trace equivalent LTGs are capable of carrying out the same sequence of actions. Consider for instance the three labeled transition graphs below.

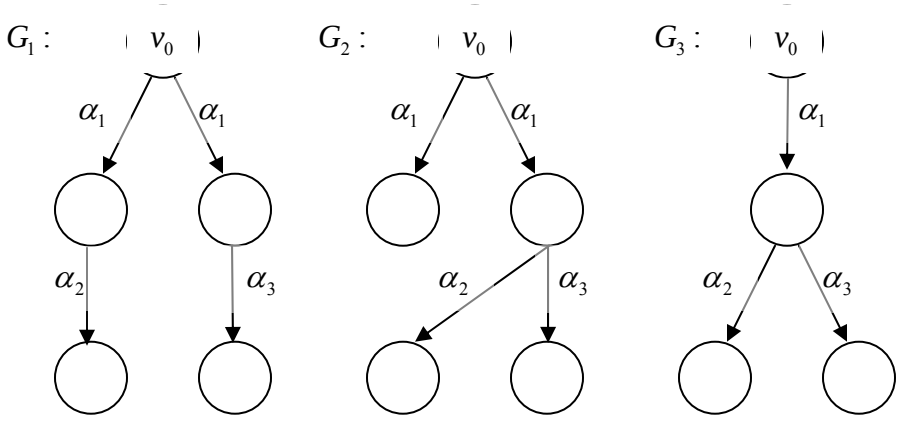



**Figure 2. Three Trace Equivalent Labeled Transition Graphs**

These three Labeled Transition Graphs are all trace equivalent.

## 2. Labeled Transition Graph Morphisms

In graph theory one fundamental equivalence relationship among graphs is called an isomorphism. An isomorphism consists of a bijective mapping between the vertices of two graphs such that each edge in one graph corresponds to an edge in the other graph and vice versa. This type of morphism is far too restrictive to be useful when trying to compare systems that are modeled using Labeled Transition Graphs. We would like to compare systems that have different numbers of system states or transitions between states. Systems that are functionally equivalent could end up having different graph representations.

Another method to determine whether two Labeled Transition Graphs are equivalent was developed by Milner [4] for use in CCS. This approach has proven to be very useful, and was used extensively throughout Bibighaus' work. This system of comparison consists of two relational mapping schemes; one called simulation, and the other referred to as bisimulation. Both schemes make use of a relation $R$ on the vertices of the two graphs. Given two Labeled Transition Graphs $G_C$ and $G_A$, $R \subseteq (V_{G_C} \times V_{G_A})$, such that *R* is Left-Right total.

**Definition 2.7: (Left-Right total mapping condition)** Given two Labeled Transition Graphs $G_A, G_C$ and relation $R$ between the vertices of $G_A$ and $G_C$, we say that *R* is Left-Right Total if:

$(\forall v : v \in V_{G_C} \Rightarrow \exists v' : v' \in V_{G_A} \wedge (v, v') \in R) \wedge (\forall v' : v' \in V_{G_A} \Rightarrow \exists v : v \in V_{G_C} \wedge (v', v) \in R)$

From here on, unless we say otherwise, we assume that all such relations are Left-Right total.

**Definition 2.8: (Simulation between LTGs)** Given two graphs $G_C, G_A$, and a relation $R \subseteq (V_{G_C} \times V_{G_A})$ we say that $G_C$ simulates $G_A$ under $R$ and write $G_C \lesssim_R G_A$ if:

$\forall v_1, v_2 \in V_{G_C} \wedge \alpha \in A_{G_C} \ni e = (v_1, \alpha, v_2) \wedge e \in E_{G_C} \exists v_1', v_2' \in V_{G_A} \ni e' = (v_1', \alpha, v_2') \wedge e' \in E_{G_A}$

**Definition 2.9: (Bisimulation between LTGs)** Given two graphs $G_C, G_A$, and a relation $R \subseteq (V_{G_C} \times V_{G_A})$ such that: $G_C \lesssim_R G_A \wedge G_A \lesssim_{R^{-1}} G_C$, we say that $G_C$ and $G_A$ are bisimilar and write: $G_C \approx_R G_A$.

Here we adopt Bibighaus' framework [2] and stipulate that the set of actions in the two graphs be the same. This simplifies our model and requires only a relation on the vertices of the two graphs. In a simulation $G_C \lesssim_R G_A$, we map the vertices from $V_{G_C}$ to vertices in $V_{G_A}$ in such a manner that we ensure all labeled edges in $E_{G_C}$ correspond to labeled edges in $E_{G_A}$. In other words, $R$ induces a relation $R_{\rightarrow} \subseteq (E_{G_C}, E_{G_A})$. In the bisimulation $G_C \approx_R G_A$, we must ensure that a mapping goes in both directions. This requires that the bisimulation relation $R$ induces relations such that $R_{\rightarrow} \subseteq (E_{G_C}, E_{G_A}) \wedge R^{-1}_{\rightarrow} \subseteq (E_{G_A}, E_{G_C})$.

Bisimulation is a more restricted form of comparison than trace equivalence. Bisimilar systems must not only agree on what they do, but also on how they do things

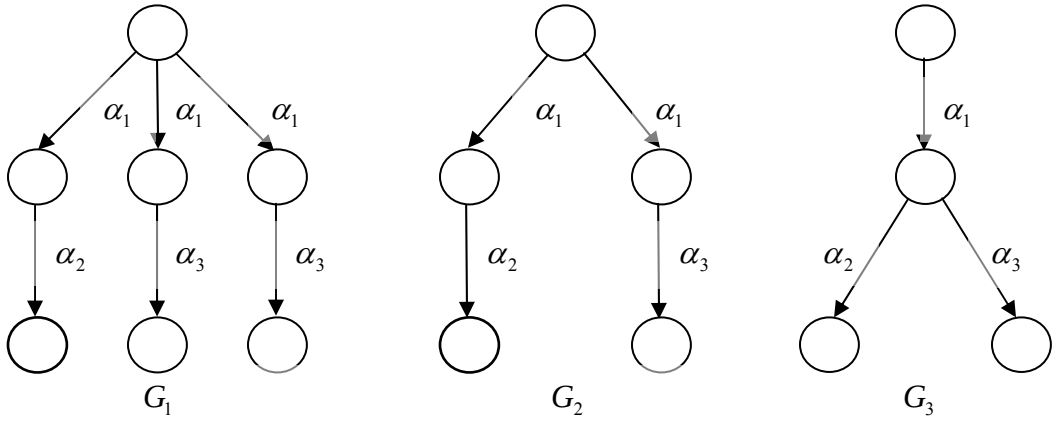



**Figure 3. Comparison Of Equivalence Relationships (Modified From [2])**

The three graphs $G_1, G_2$, and $G_3$ shown in figure 3 are all trace equivalent, yet only $G_1$ and $G_2$ are bisimilar. Moller and Smolka [8] rather elegantly explain bisimulation in terms of a game involving two processes *(P, Q)*, whereby two players alternate moves in the following manner: Player 1 chooses a transition of one of the processes, and player 2 must subsequently respond by selecting an identical transition in the other process. Then

they reverse roles and player 2 gets to choose a transition in either *P* or *Q*, and player 1 must find an identical transition in the other process. If one of the players is ever unable to respond to the adversary's move, he loses the game. In this case the systems are not bisimilar.

**Lemma 2.1: Bisimulation implies trace equivalence**.

$G_x \approx_R G_y \Rightarrow G_x \overset{trace}{\equiv} G_y$

**Proof:** Suppose by way of contradiction that there are two Labeled Transition Graphs $G_x$ and $G_y$ such that $G_x \approx G_y$ under the relation $R$ and $T(G_x) \neq T(G_y)$. From Definition 2.9 we know that $G_x \lesssim_R G_y \wedge G_y \lesssim_{R^{-1}} G_x$. Since $T(G_x) \neq T(G_y)$, there must be at least one trace *t* in one graph that is not present in the other. Without loss of generality, let $t \in T(G_x) \wedge t \notin T(G_y)$. By Definition 2.3, $t = \langle a_n \rangle$. Since $t \notin T(G_y)$, there is at least one action $a_i \in a_n$ such that the relation R is unable to ensure $R \subseteq (E_{G_x}, E_{G_y})$. This however, this violates the bisimulation conditions set forth in Definition 2.9, which in turn implies that $\neg(G_x \approx_R G_y)$. Since this contradicts our initial assumption, we conclude that $G_x \approx_R G_y \Rightarrow G_x \overset{trace}{\equiv} G_y$.

Often, bisimulation is the preferred method of testing whether two graphs are equivalent. Bisimulation is not only more discriminating, but surprisingly has been shown to be less computationally complex than trace equivalence. Paige and Tarjan [9] have developed an algorithm for determining whether or not two systems are bisimilar that runs in $O(m \log n)$ time, where *n* is the number of states and *m* is the branching factor. Moller and Smolka [8], on the other hand, have shown that under certain conditions proving trace equivalence is PSPACE-compete.

## 3. Equivalence Classes of Labeled Transition Graphs

We now show that bisimulation forms an equivalence relation on a set of Labeled Transition Graphs.

**Lemma 2.2: Bisimulation is reflexive**

**Proof**: From Definition 2.9 we have that:

$$G_A \approx G_A \Leftrightarrow (G_A \lesssim_R G_A) \wedge (G_A \lesssim_{R^{-1}} G_A)$$

Letting $R = I$, where $I$ is the identity relation and $I^{-1} = I$, yields:

$$(G_A \lesssim_I G_A) \wedge (G_A \lesssim_{I^{-1}} G_A) \Rightarrow G_A \approx_I G_A .$$

**Lemma 2.3: Bisimulation is symmetric**

$$G_A \approx_R G_B \Leftrightarrow G_B \approx_{R^{-1}} G_A$$

**Proof:** Suppose $G_A \approx_R G_B$. Then by Definition 2.9 we have that $G_A \lesssim_R G_B$ and $G_B \lesssim_{R^{-1}} G_A$. Furthermore, in order to have $G_B \approx_{R'} G_A$ there must exist a relation $R'$ such that $G_B \lesssim_{R'} G_A$ and $G_A \lesssim_{R'^{-1}} G_B$. Letting $R' = R^{-1}$ we get $G_B \lesssim_{R^{-1}} G_A$ and since $(R')^{-1} = (R^{-1})^{-1} = R$ we also have $G_A \lesssim_R G_B$.

**Lemma 2.4: Bisimulation is transitive**

$$(G_A \approx_{R_1} G_B) \wedge (G_B \approx_{R_2} G_C) \Rightarrow \exists R \ni G_A \approx_R G_C$$

**Proof:** Suppose we have $G_A \approx_{R_1} G_B$, under relation $R_1$, $G_B \approx_{R_2} G_C$, and $G_A \not\approx_R G_C$. Then according to Definition 2.9 we have:

$$(G_B \lesssim_{R_1} G_A) \wedge (G_A \lesssim_{R_1^{-1}} G_B) \text{ and } (G_C \lesssim_{R_2} G_B) \wedge (G_B \lesssim_{R_2^{-1}} G_C)$$

Since the bisimulation relations are by definition Left-Right total, they can be composed. Letting $R_3 = R_2 \circ R_1$ and $R_3^{-1} = R_1^{-1} \circ R_2^{-1}$, and using these new relations we then have : $(G_C \lesssim_{R_3} G_A) \wedge (G_A \lesssim_{R_3^{-1}} G_C)$, which by Definition 2.9 means that:

$G_A \approx_{R_3} G_C$. We have therefore demonstrated that:

$(G_A \approx_{R_1} G_B) \wedge (G_B \approx_{R_2} G_C) \Rightarrow \exists R \ni G_A \approx_R G_C$.

**Theorem 2.1: Bisimulation forms an equivalence relation on a set of Labeled Transition Graphs**

**Proof:** This follows directly from Lemma 2.2, Lemma 2.3, and Lemma 2.4.

An important consequence of Theorem 2.1 is that a set of Labeled Transition Graphs can be partitioned into equivalence classes, which we call Bisimulation classes and denote by $\tilde{\approx}G$. Every pair of graphs in a bisimulation class are bisimilar and any two graphs in different bisimulation classes are not bisimilar. Therefore, if we wish to prove that a given graph, not contained in a particular bisimulation class, is bisimilar to a specific graph within that bisimulation class, it is sufficient to prove that it is bisimilar to any graph in the bisimulation class. This turns out to be a very useful property since it oftentimes can be very difficult to prove bisimilarity between graphs that have greatly varying degrees of complexity. We may be able to simplify our proof by moving laterally rather than horizontally along the complexity scale, and instead attempt to prove bisimilarity between graphs that contain similar degrees of complexity.

In this chapter we briefly discussed Process Algebras and explained why we have chosen to work with the operational semantic framework of Bibigihaus. We then introduced the notion of a Labeled Transition Graph. This discussion covered the notion of a trace in a Labeled Transition Graph, as well as when two LTGs are considered to be trace equivalent. Subsequently we introduced simulation and bisimulation morphisms

between Labeled Transition Graphs. Finally, we proved that bisimulation forms an equivalence relation on a set of Labeled Transition Graphs, and discussed how this fact could be useful when trying to establish a bisimulation relation between two Labeled Transition Graphs.

# III. ABSTRACTION AND REFINEMENT

## A. OVERVIEW

In this chapter we introduce the concepts of abstraction and refinement and show how they are used in Computer Science. We then discuss basic refinement of Labeled Transition Graphs as well as two other forms of refinement, called vertex refinement and edge refinement.

Computers are very complex machines. In order to study them, computer scientists must leverage the principle of abstraction. Abstraction is a process whereby one reduces the complexity of a system by removing detail. By doing so, one can focus on a few important concepts at a time. For example, the typical view of computer architecture is as a series of abstraction layers as depicted below:

| | | |
|---|---|---|
| Level 6 | User | Executable Programs |
| Level 5 | High-level Language | C, C++, Java, Fortran |
| Level 4 | Assembly Language | Assembly Code |
| Level 3 | System Software | Operating System |
| Level 2 | Machine | Instruction Set Architecture |
| Level 1 | Control | Microcode or Hardwired |
| Level 0 | Digital Logic | Circuits , Logic Gates etc |

Table 1. **The Abstract Layers Of A Computer System (Modified From [10])**

Each of the above layers represents a particular view of the system. Given a particular sequence of actions, we can model what is taking place at each of these layers with a Labeled Transition Graph. An abstraction is a mapping between two representations where we strip away extraneous details from one system, retaining only the properties pertinent to a higher level view of the system. The inverse of this mapping is referred to as a refinement. In terms of Labeled Transition Graphs, we add detail in the form of vertices and edges to a graph to obtain a more detailed representation of the system.

## B. BASIC REFINEMENT

Refinement mappings are of particular interest since we typically build computer systems and design software in a top down fashion; beginning first with an abstract specification and then developing a concrete implementation. [11] Throughout this process we need to ensure that our implementation preserves all of the desired properties of our abstraction.

**Definition 3.1: (A Labeled Transition Graph refinement)** Given two LTGs $G_C$ and $G_A$ we say that $G_C$ is a refinement of $G_A$ if and only if $G_C \lesssim_R G_A$.

**Theorem 3.1: Graph refinement implies trace containment**

$$\forall p_C \in P(G_C)\ \exists p_A \in P(G_A) \ni T(p_C) = T(p_A) \wedge G_C|_{p_C} \lesssim_R G_A|_{p_A}$$

**Proof:** We prove this using induction on the length of the path. Suppose there are two Labeled Transition Graphs $G_C$ and $G_A$ such that $G_C \lesssim_R G_A$. We use the notation $p_A$ and $p_C$ to represent directed paths in $G_A$ and $G_C$ respectively.

Base case: Let $p_C \in P(G_C)$ be a directed path of length 0. Since a trivial walk is a part of any graph, we know that there exists a path $p_A$ of length zero in $G_A$. From Definition 2.4 we know that this implies that the empty trace is part of each graph's trace set, which in turn means that the theorem is true for the base case.

Induction: Let |p| represent the length of a path p. We then assume that $\forall p, |p| \leq n$, the theorem is true. Let $p_C \in P(G_C), |p_C| = n+1$, $p_C = < e_1, e_2, ..., e_n, e_{n+1} >$, and $P_C' = < e_1, e_2, ..., e_n >$. Then by induction $\exists p_A' = < e_1', e_2', ...e_n' > \ni T(p_A') = T(p_C')$. Let $e_n = < v_n, \alpha_n, v_{n+1} >$, $e_n' = < v_n', \alpha_n, v_{n+1}' >$, and $e_{n+1} = (v_{n+1,} \alpha_{n+1}, v_{n+2})$. Since $G_C \lesssim_R G_A$, $\exists v_{n+2}' \in V_{G_A}$ and an edge $(v_{n+1}', \alpha_{n+1,} v_{n+2}') \ni (v_{n+2}, v_{n+2}') \in R$. We have therefore constructed, a directed path $P_A$ in $G_A$ such that $T(p_A) = T(p_C)$.

This theorem tells us that, given two graphs $G_A$ and $G_C$, where $G_C$ is a refinement of $G_A$, the refinement relation *R* ensures that $G_A$ contains all of the directed paths of $G_C$, and hence also contains $G_C$'s trace set.

To illustrate abstraction and refinement we revisit our flashlight LTG example. We begin by questioning whether or not our model was correct. We correctly identified the binary nature of the illumination property, but made assumptions about the flashlight battery and light bulb. Clearly, actions such as the light bulb burning out, or the battery running out of electrical charge, would lead to failures. In this case the flashlight would remain off, regardless of how many times we pushed the "on" button. We can rectify our oversight by adding additional states and actions to our model as depicted below.

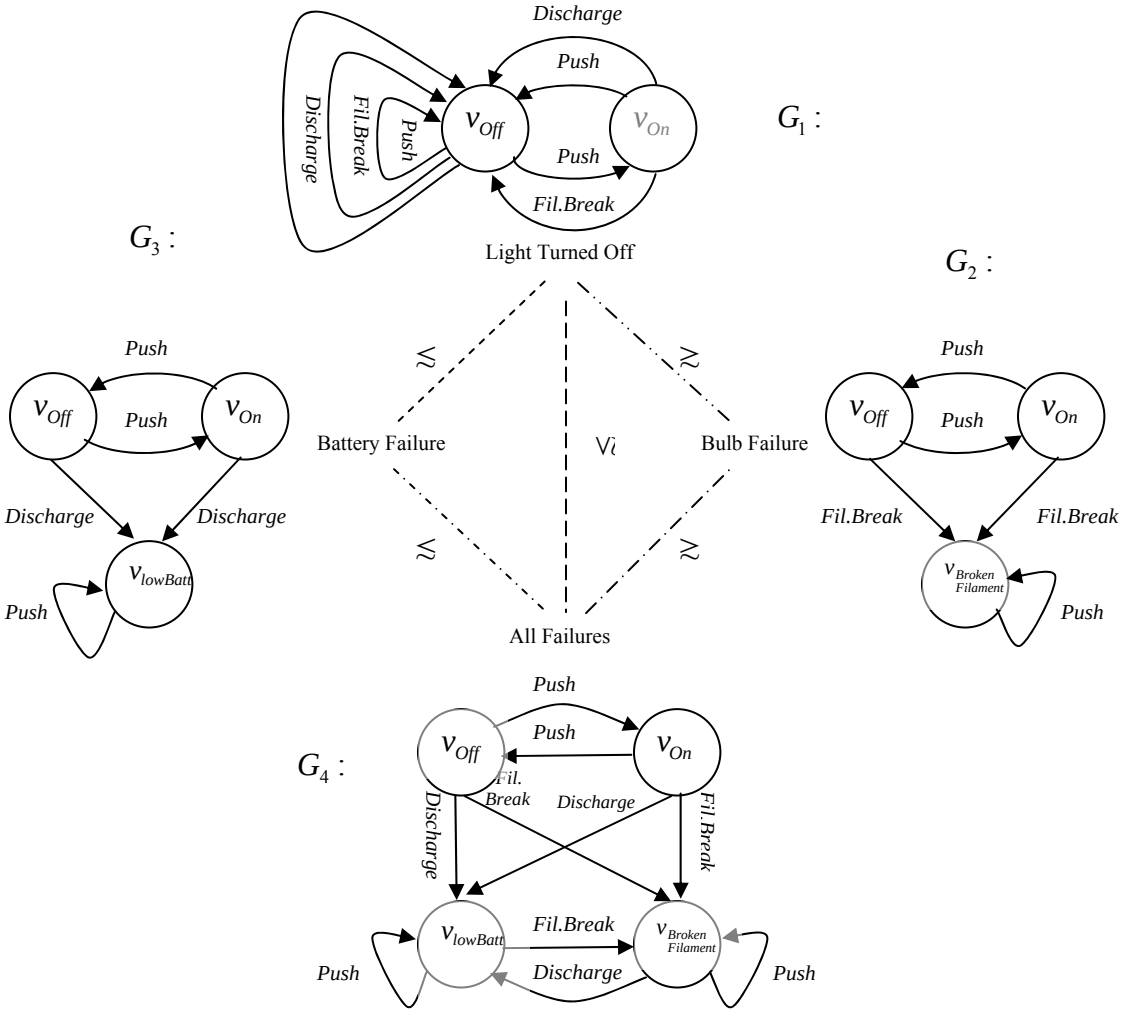



**Figure 4. Flashlight LTG Refinements**

In $G_2$ we added a broken filament vertex along with a *Fil.Break* action, whereas in $G_3$ we added a low battery vertex and a *Discharge* action. In $G_4$ we added both of these new states and their associated labeled edges. Insofar, $G_4$ is an abstraction for both $G_2$ and $G_3$, since it contains all of the directed paths and thus the traces of $G_2$ and $G_3$. Likewise, $G_1$ is an abstraction of the three other graphs. The simulation morphisms map $v_{LowBatt}$ and $v_{\substack{Broken \\ Filament}}$ vertices of $G_2$, $G_3$, and $G_4$ to the $v_{Off}$ vertex in $G_1$. Although $G_1$ contains the same vertices as our initial model, it now has quite a few additional labeled edges. These labeled edges account for all of the situations we failed to consider in our original model.

It may initially seem rather odd that $G_1$ is non-deterministic with regard to the *Push* transition from the $v_{Off}$ vertex. Yet, from a practical standpoint this is not so surprising. Anyone who has ever attempted to use an old flashlight, can attest to the uncertainty regarding what effect switching it on will have. From a theoretical point of view however, one will notice that $G_4$ is deterministic while still being able to model the desired property. Oftentimes, as in this case, non-determinism can occur in a model due to under specification.

We proceed by proving the following lemmas and theorems regarding simulation and refinement:

**Lemma 3.1: Simulation is reflexive.**

$G_A \lesssim_I G_A \Rightarrow G_A \approx_I G_A$

**Proof:** Let $G_A$ be a Labeled Transition Graph and let *I* be the identity mapping on the vertices of $G_A$. Then we have: $G_A \lesssim_I G_A$. Since $I^{-1} = I$, we also have:

$G_A \lesssim_{I^{-1}} G_A$. Applying Definition 2.9 yields $G_A \lesssim_I G_A \Rightarrow G_A \approx_I G_A$.

**Lemma 3.2: Simulation is transitive.**

$(G_B \lesssim_{R_1} G_A) \wedge (G_C \lesssim_{R_2} G_B) \Rightarrow \exists R_3 \ni G_C \lesssim_{R_3} G_A$

Proof: Suppose we have three Labeled Transition Graphs $G_A, G_B$, and $G_C$ such that $(G_B \lesssim_{R_1} G_A) \wedge (G_C \lesssim_{R_2} G_B)$. $R_3$ such that $R_3 = R_2 \circ R_1$. If $(v_1, \alpha, v_2) \in E_{G_C}$ then there exists $v_1', v_2' \in V_{G_B}$ such that $(v_1, v_1'), (v_2, v_2') \in R_2$, and $(v_1', \alpha, v_2') \in E_{G_B}$. Likewise, we have $(v_1', v_1''), (v_2', v_2'') \in R_1$, where $v_1'', v_2'' \in V_{G_A}$. Therefore by the relation composition there exist a relation $R_3$ such that $R_3 = R_2 \circ R_1$ and $(v_1, v_1''), (v_2, v_2'') \in R_3$, and $(v_1'', \alpha, v_2'') \in E_{G_A}$ and we have shown that simulation is transitive.

**Theorem 3.2: Simulation forms a preorder on a set of LTG refinements**

**Proof:** This follows directly from Lemma 3.1, and Lemma 3.2.

## C. OTHER FORMS OF REFINEMENT

Other forms of refinement have been suggested. In this section we present two of the most notable among them.

### 1. Vertex Refinement

Derrick and Boiten [12, 13] introduced weak refinement. Here internal (hidden) transitions are added to an abstract graph. We refer to this type of refinement as Vertex Refinement.

**Definition 3.2: (Vertex refinement)** Given two Labeled Transition Graphs $G_A$ and $G_C$, such that there exists a vertex $v \in V_{G_A}$ and $v$ is refined by a set of vertices $V \subseteq V_{G_C}$, we write $V \dot{<} v$.

To illustrate the process, consider the vertex $v_{Off}$ in the Labeled Transition Graph $G_1$ from Figure 4. Here the actions, *Fil.Break*, and *Discharge*, do not result in a change of state. The LTG $G_4$ on the other hand, captures internal changes in $v_{Off}$ caused by these actions by adding two additional vertices $v_{lowBatt.}$ and $v_{\substack{Broken \\ Filament}}$. These new states both map to the $v_{Off}$ vertex in $G_1$ as depicted in the figure below.

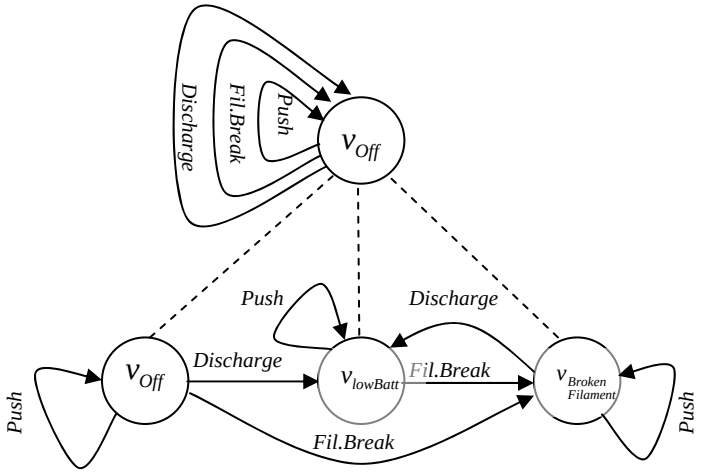



**Figure 5. Vertex Refinement**

### 2. Edge Refinement

Additionally, Derrick and Boiten [12, 13], along with Bossi, Piazza and Rossi [14] developed a refinement process known as Non-Atomic or Action refinement. Here a single labeled edge is replaced by a Labeled Transition Graph. We refer to this type of refinement as Edge refinement.

**Definition 3.3: (Edge refinement)** Given two Labeled Transition Graphs $G_A$ and $G_C$, such that there exists an edge $e \in E_{G_A}$ and $e$ is refined by $G_C$ we write $G_C \vec{<} e$.

An example of when such a refinement may be appropriate, can be found in the modeling of the dialing of a phone, as depicted by the LTGs $G_A$ and $G_C$ in Figure 6.

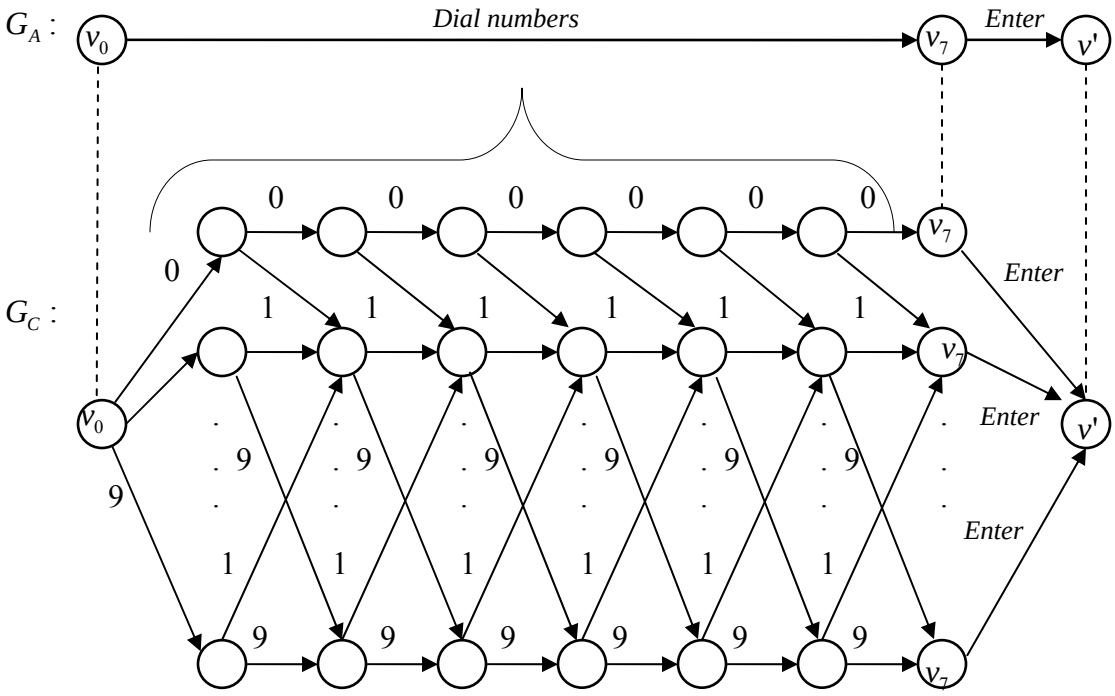



**Figure 6. Edge Refinement**

In the abstract Labeled Transition Graph $G_A$ we model the dialing process with a single edge labeled *Dial numbers*, whereas the refinement $G_C$ displays all of the edges and vertices associated with the process of dialing the number. This edge refinement requires a mapping relation between the labeled edge in $G_A$ and a sub-graph of $G_C$. Using the notation for a transition introduced in Definition 2.1, and representing each of the seven digit selection actions as $digit_i$, where $digit_i \in \{0, 1, 2, ..., 9\}$ and $i \in \{1, 2, 3...,7\}$, we write the transition or sequence of transitions leading to vertex $v_7$ for the respective LTGs as:

$$Dial\ numbers(v_0) = v_7$$

$$digit_7(digit_6(digit_5(digit_4(digit_3(digit_2(digit_1(v_0))))))) = v_7$$

Doing so allows us to compare the transition in $G_A$ with the edge refinement in $G_C$, and aids in our reasoning about the vertices of the two representations. Both models contain $v_0$ and $v_7$. The initial vertex $v_0$ contains no information concerning the phone number,

whereas $v_7$ includes all seven digits of the phone number. In $G_A$ the action *dial numbers* on $v_0$ leads directly to $v_7$. This is analogous to hitting speed dial or re-dial on your phone. On the other hand, in $G_C$ the digits must be entered one at a time. Consequently, the intermediate vertices contain successively more information regarding the phone number to be dialed. For instance:

$digit_1(v_0) = v_1$, where $v_1$ contains the first digit of the phone number.

$digit_2(digit_1(v_0)) = v_2$, where $v_2$ contains the first two digits of the phone number.

$digit_3(digit_2(digit_1(v_0))) = v_3$, where $v_3$ contains three digits of the phone number.

Etc…

Furthermore, it now becomes clear that the composition of the actions in $G_C$ results in the same outcome as the *dial numbers* action in $G_A$, and that the intermediate states in $G_C$ are internal to the $v_7$ state in $G_A$. What's more, although this example was deterministic, it is worth noting that if the transition in the abstract Labeled Transition Graph is non-deterministic one or more of the intermediate transitions in the edge refinement will be non-deterministic as well.

## D. SUMMARY

In this chapter we presented abstraction and refinement in the general context of Computer Science, and then more specifically in terms of Labeled Transition Graphs. We first explained that abstraction is important, particularly when dealing with complex systems, because it allows us to factor out extraneous information and focus on a small number of properties of interest. We then discussed the top down approach typically seen in secure hardware and software development, and emphasized refinement's role in ensuring that implementations retain the desired properties stipulated in the abstract design specification.

Subsequently we defined Labeled Transition Graph refinement as follows: Given two Labeled Transition Graphs $G_A$ and $G_C$, if $G_C$ is able to simulate $G_A$ we said

it was a refinement of $G_A$. Moreover, we showed that if this was the case, all of the traces in $G_C$ will be contained in $G_A$. Additionally we proved that a set of Labeled Transition Graph refinements forms a preorder. Finally, we introduced two special types of refinements called Vertex and Edge refinements.

THIS PAGE INTENTIONALLY LEFT BLANK

# IV. DOUBLY LABELED TRANSITION GRAPHS (DLTGS)

## A. INTRODUCTION

In this chapter we introduce Doubly Labeled Transition Graphs and a new concept, Modal Constrained Bisimulation. We provide an example using the non-interference security property that shows that Labeled Transition Graphs are inadequate when trying to ensure that a refinement preserves all of the desired security properties. Finally, we prove some properties related to Modal Constrained Bisimulation, and discuss edge refinement in Doubly Labeled Transition Graphs.

As we have seen, an abstract Labeled Transition Graph defines an upper bound on a set of LTG refinement behaviors. The simulation morphism between the graphs ensures that any trace in a refinement of the graph is also present in the abstract graph. Thus, no new behavior is introduced during the refinement process. Based on Larsen[15] and Dams' work [16], Schmidt [5] and Bibighaus [2] subsequently extended Labeled Transition Graphs by introducing an additional modal edge labeling scheme. In this Doubly Labeled Transition Graph (DLTG) model, each edge contains an additional must □, or may ◊ label.

**Definition 4.1: (Doubly Labeled Transition Graph)** Let *G* be a directed labeled transition graph defined by the n-tuple $G = \langle V, \mathrm{A}, E^{\diamond}, E^{\square}, v_0 \rangle$ where

$V$ is a set of vertices called states.

A is a set of action labels.

$E^{\diamond}$ is the set of doubly labeled "May" edges.

$E^{\square}$ is the set of doubly labeled "Must" edges, such that $E^{\square} \subseteq E^{\diamond}$

$v_0$ is a distinguished start vertex such that $v_0 \in V$.

We introduce the second set of Must and May labels to help us distinguish between edges in the abstract graph that must be present in any refinement, and edges in the abstract graph that may or may not exist in a refinement. For the sake of consistency, we require that a "Must" edge also is considered to be a "May" edge: That is to say, if an edge is labeled with the Must element it is also included in the May elements . We use the following shorthand notation to denote an edge of a DLTG $G(v_1, \alpha, \mu, v_2)$, where $v_1, v_2 \in V_G$ , $\alpha \in A_G$, and $\mu \in \{\Diamond, \Box\}$

## B. MODAL CONSTRAINED BISIMULATION (MCB)

We leverage this new ability to distinguish between edges in a graph by defining a new type of graph morphism between DLTGs.

**Definition 4.2: (The Must subgraph of a DLTG)**

Given a Doubly Labeled Transition Graph *G*. Let $G^{[]}$ denote the induced subgraph with respect to the □ labeled edges of *G*.

**Definition 4.3: (The Must image in a DLTG refinement)**

Given two Doubly Labeled Transition Graphs $G_A$ and $G_C$, where $G_A^{[]}$ and $G_C^{[]}$ represent the respective Must Subgraphs of $G_A$ and $G_C$, and *R* is a relation such that $G_C \lesssim_R G_A$. We denote the subgraph of $G_C^{[]}$ that is restricted to the image of $G_A^{[]}$ under the relation mapping $R^{-1}$ as: $G_C^{[]}|_{R^{-1}(E_{G_A}^{[]})}$.

**Definition 4.4: (Modal Constrained Bisimulation between DLTGs)**

If there exist two Doubly Labeled Transition Graphs $G_A$ and $G_C$ and Left-Right total relation $R \subseteq (V_{G_A} \times V_{G_C})$, such that: $G_C \lesssim_R G_A$ and $G_A^{[]} \approx_R G_C^{[]}|_{R^{-1}(E_{G_A}^{[]})}$, then there exists a Modal Constrained Bisimulation morphism between the two graphs, and we write $G_A \not\approx_R G_C$.

Note that Modal Constrained Bisimulation only requires that $G_A^{[]} \approx_R G_C^{[]}|_{R^{-1}(E_{G_A}^{[]})}$. This allows for additional □ (Must) labeled edges to be added to the refinement. Since we have $G_C \lesssim_R G_A$, these added □ edges will map back to the abstract graph under the relation *R*. However, it is not required that they map to the $G_A^{[]}$ subgraph. In the case where no additional □ edges are added to the refinement, or the additional □ edges that are added all map to $G_A^{[]}$, the situation is simplified and we then end up with $G_C \lesssim_R G_A$ and $G_A^{[]} \approx_R G_C^{[]}$.

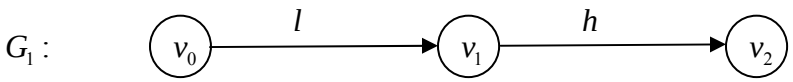
Modal Constrained Bisimulation not only guarantees that any trace in a refinement is present in the abstract graph, but it also ensures that all of the must transitions stipulated in the abstract graph are faithfully replicated in the refinement. The Doubly Labeled Transition Graphs along with Modal Constrained Bisimulation establishes a lower bound on the set of behaviors of a graph refinement. In fact, Schmidt [5] has shown that Doubly Labeled Transition Graph refinements are capable of preserving a larger subset of properties than Labeled Transition Graphs.

## C. THE MOTIVATION BEHIND DLTGS

To highlight the benefit of using Doubly Labeled Transition Graphs, we present the following example from Dinolt [17]: Suppose we wanted to develop a Multi-Level Computer System (MLS) that is capable of supporting both high and low level users. Then in order for the system to be secure, we must ensure that low users are never given access to high level data, nor are even aware of the fact that high level instructions are being carried out on the system. In other words, from a low level user's perspective, we require that the system behave exactly like a low level system. In essence then, we have defined a non-interference security property that we want our system to uphold.

We can model such a system using Labeled Transition Graphs by adopting a Goguen-Mesquer type Security Model [18]. We start with a set of high and low users and an LTG that represents our MLS. The actions of the graph now represent commands

issued by users, and are therefore designated as either high *h*, or low *l*. Additionally, we define an *Out* function that returns the output that is visible to a user after a set of actions has been executed by the system. Having done so, it now becomes necessary for us to also define a *Purge* function that will remove all of the high actions from a low user's visible output. Consider the traces and purged traces of the following graphs:

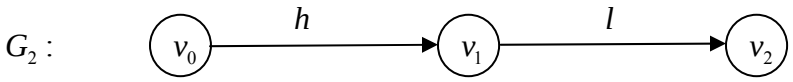



$$T(G_1) = \{<>, < l >, < l, h >\}$$

$$Purge(T(G_1)) = \{<>, < l >, < l >\}$$

**Figure 7. A Secure Trace With Respect To Non-Interference**

The graph $G_1$ is secure, since the *Purge* of each of the traces corresponds to low actions and each of these low actions are part of the Trace set of the graph.




$$T(G_2) = \{<>, < h >, < h, l >\}$$

$$Purge(T(G_2)) = \{<>, <>, < l >\}$$

**Figure 8. A Non-Secure Trace With Respect to Non-Interference**

However, the graph $G_2$, is not secure. Here the *Purge* results in a trace $< l >$, that is not part of the set of traces in $G_2$. If a low user saw this trace, he or she could surmise that a high action had taken place. To remedy the situation we can add an additional edge $(v_0, l, v_1)$, as depicted in the graph $G_3$.

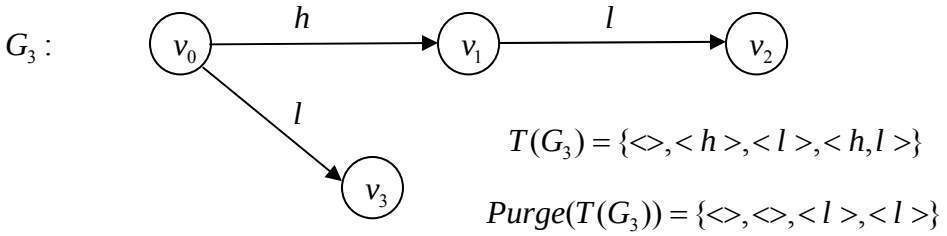



**Figure 9. Ensuring Trace Security With Respect To Non-Interference**

As a result of the addition of this edge, all traces produced by the Purge function now correspond to traces in $G_3$'s trace set, and $G_3$ is now secure.

There are two important observations to be made from this example. First and foremost, Labeled Transition Graphs are not sufficient when it comes to modeling security properties. Both $G_1$ and $G_2$ are valid refinements of the Labeled Transition Graph $G_A$ depicted below,

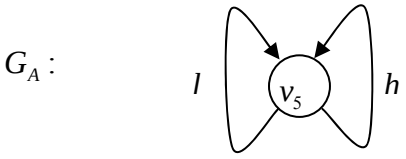



**Figure 10. The Abstract Non-Interference Labeled Transition Graph**

yet, $G_2$ is not a secure.

Secondly, in order to guarantee that the non-interference property is retained in LTG refinements, we must not only ensure that all traces of the refinement, map to traces in the abstract system, but we must also ensure that a subset of the traces of the Abstract graph map to traces in the refinement graph. This is exactly what Doubly Labeled Transition Graphs allow us to do. In this case, we could have ensured that the non-interference security property was upheld by using a DLTG and requiring that the low transitions are designated as Must transitions.

Having demonstrated the utility of Doubly Labeled Transition Graphs, we proceed by proving some properties related to Modal Constrained Bisimulation.

**Lemma 4.1 MCB is reflexive**

$G_A \approx_I G_A$

**Proof:** Let $G_A$ be a Labeled Transition Graph and let *I* be the identity mapping on the vertices of $G_A$. Then according to Definition 4.3 we have: $G_A \lesssim {}_I G_A$ and $G_A^{[]} \approx_I G_A^{[]}|_{I^{-1}(E_{G_A}^{[]})}$. However, in this case $G_A^{[]}|_{I^{-1}(E_{G_A}^{[]})} =_I G_A^{[]}$, and we have that $G_A^{[]} \approx_I G_A^{[]}$. Furthermore, since $I^{-1} = I$ we also have $G_A \lesssim_{I^{-1}} G_A$. Applying Definition 2.9 yields $G_A \approx_I G_A$.

**Lemma 4.2 MCB is transitive**

$(G_A \approx_{R_1} G_B) \wedge (G_B \approx_{R_2} G_C) \Rightarrow \exists R \ni G_A \approx_R G_C$

**Proof:** Suppose we have three Doubly Labeled Transition Graphs $G_A, G_B$, and $G_C$ such that $G_A \approx_{R_1} G_B$ and $G_B \approx_{R_2} G_C$. Then by applying Definition 4.4 we have that:

$$(G_B \lesssim_{R_1} G_A) \wedge (G_A^{[]} \approx_{R_1} G_B^{[]}|_{R_1^{-1}(E_{G_A}^{[]})})$$

$$(G_C \lesssim_{R_2} G_B) \wedge (G_B^{[]} \approx_{R_2} G_C^{[]}|_{R_2^{-1}(E_{G_B}^{[]})})$$

From Definition 2.9 we know that:

$$(G_A^{[]} \approx_{R_1} G_B^{[]}|_{R_1^{-1}(E_{G_A}^{[]})}) \Rightarrow (G_B^{[]}|_{R_1^{-1}(E_{G_A}^{[]})} \lesssim_{R_1} G_A^{[]}) \wedge (G_A^{[]} \lesssim_{R_1^{-1}} G_B^{[]}|_{R_1^{-1}(E_{G_A}^{[]})}$$

$$(G_B^{[]} \approx_{R_2} G_C^{[]}|_{R_2^{-1}(E_{G_B}^{[]})}) \Rightarrow (G_C^{[]}|_{R_2^{-1}(E_{G_B}^{[]})} \lesssim_{R_2} G_B^{[]}) \wedge (G_B^{[]} \lesssim_{R_2^{-1}} G_C^{[]}|_{R_2^{-1}(E_{G_B}^{[]})}$$

Using reasoning similar to that in Lemma 3.2, we proceed as follows: Suppose $v_1, v_2, v_3, v_4 \in V_{G_C}$ and that $(v_1, \alpha_1 \diamond, v_2) \in E_{G_C}^{\diamond}$ and $(v_3, \alpha_2, \Box v_4) \in E_{G_C}^{[]}$ Then there exists $v_1', v_2', v_3', v_4' \in V_{G_B}$ such that $(v_1, v_1')$, $(v_3, v_3')$, $(v_2, v_2')$, $(v_4, v_4') \in R_2$,

$(v_3', v_3), (v_4', v_4) \in R_2^{-1}$, and $(v_1', \alpha_1 \diamond, v_2') \in E_{G_B}^{\diamond} \wedge (v_3', \alpha_2, \square, v_4') \in E_{G_B}^{[]}$. Likewise, if $v_1', v_2', v_3', v_4' \in V_{G_B}$, then there exists $v_1'', v_2'', v_3'', v_4'' \in V_{G_A}$ such that

$(v_1', v_1''), (v_3', v_3''), (v_2', v_2''), (v_4', v_4'') \in R_1$, $(v_3'', v_3')$, $(v_4', v_4'') \in R_1$, and

$(v_1'', \alpha_1 \diamond v_2'') \in E_{G_A}^{\diamond} \wedge (v_3'', \alpha_2, \square, v_4'') \in E_{G_A}^{[]}$. Therefore by the relation composition $R_3 = R_2 \circ R_1$ and $R_3^{-1} = R_1^{-1} \circ R_2^{-1}$ we have:

$(v_1, v_1''), (v_2, v_2''), (v_3, v_3''), (v_4, v_4'') \in R_3, (v_3'', v_3), (v_4'', v_4) \in R_3^{-1}$, and

$(v_1'', \alpha_1 \diamond, v_2'') \in E_{G_A}^{\diamond} \wedge (v_3, \alpha_2, \square, v_4) \in E_{G_C}^{[]}$. Thus we have shown that:

$(G_C \precsim_{R_3} G_A) \wedge (G_A^{[]} \approx_{R_3} G_C^{[]}|_{R_3^{-1}(E_{G_A}^{[]})}) \Rightarrow G_C \models\!\approx_{R_3} G_A$ .

**Theorem 4.1 MCB forms a preorder on a set of DLTG refinements**

**Proof**: This follows directly from Lemma 4.1 and Lemma 4.2

## D. REFINEMENT OF DLTGS

The edge set of a Doubly Labeled Transition Graph contains two subsets, namely $E^{[]}$ and $E^{\diamond}$. It therefore is necessary to specify a couple of additional requirements regarding refinement for a Doubly Labeled Transition Graph.

**Definition 4.5: (Basic refinement consistency condition for DLTGs)**

Given two DLTGs $G_A$ and $G_C$ such that $G_A \models\!\approx_R G_C$, we say that $R$ satisfies the basic refinement condition if : $\forall e \in E_{G_A}^{[]} \wedge \forall e' \in E_{G_C}$ ∋ $(e', e) \in R \Rightarrow e' \in E_{G_C}^{[]}$

The basic refinement Consistency Condition for DLTGs ensures that all "Must" edges specified in the abstract graph persist throughout multiple refinements. A similar consistency argument exists in the case of edge refinement and results in the following theorem:

**Theorem 4.2 Edge refinement theorem.**

Let $G_A \lessapprox_{R_1} G_B$, $e \in E_A^{[]}$, $G_B'$ be the edge refinement of $e$ in $G_B$, then $E_{G_B'}^{[]} = E_{G_B'}^{<>}$.

(Every edge of a graph that edge-refines a "Must" edge of a DLT, also is a "Must" edge.)

**Proof**: In addition to $G_A \lessapprox_{R_1} G_B$, suppose, by way of contradiction, that we have an additional DLTG $G_C$ such that $G_B \lessapprox_{R_2} G_C$ and $\exists e' \in E_{G_B'}^{<>} \wedge e' \notin E_{G_B'}^{[]}$. Then according to Definition 4.4 either $\exists e'' \ni (e', e'') \in R_2$ or $\nexists e'' \ni (e', e'') \in R_2$. If $\nexists e'' \ni (e', e'') \in R_2$ then there can not exist a relation such that $G_A^{[]} \approx {}_R G_C^{[]}|_{R^{-1}(E_{G_A}^{[]})}$. However, since $G_A \lessapprox_{R_1} G_B$ and $G_B \lessapprox_{R_2} G_C$ we know from Definition 4.4 that $G_A^{[]} \approx_{R_1} G_B^{[]}|_{R_1^{-1}(E_{G_A}^{[]})}$ and $G_B^{[]} \approx_{R_2} G_C^{[]}|_{R_2^{-1}(E_{G_A}^{[]})}$. Furthermore from Lemma 4.2 we know that there exists a relation $R$ such that $G_A \lessapprox_R G_C$, which in turn means that $G_A^{[]} \approx {}_R G_C^{[]}|_{R^{-1}(E_{G_A}^{[]})}$. However, this results in a contradiction, and we therefore conclude that in order to have a sequence of DLTG refinements we must ensure that every edge of a graph that edge-refines a "Must" edge, also is a "Must" edge.

To illustrate the point further, consider the DLTG $G_A$ of a soda machine borrowed from Bibighaus' work.[1] This simple machine accepts a coin and dispenses a soda. The coin edge is represented by a ◊ edge, because as many of us unfortunately know, vending machines won't always accept our change. The soda edge, on the other hand, is depicted as a □ edge. This is because once the soda machine has accepted payment; we want it to faithfully dispense a soda to the consumer. Suppose now that we edge-refine $G_A$ as depicted in the graph $G_B$, but that we inadvertently classify one of the edges as a ◇ edge. Since the refinement process is not complete we subsequently create another graph $G_C$ that includes yet more detail. However, to the chagrin of our grape soda patrons, we

[1] See Bibighaus [2], p. 6.

did not require that the "*Dispense Soda*" edge had to appear below. This oversight results in a failure in the DLTG refinement.

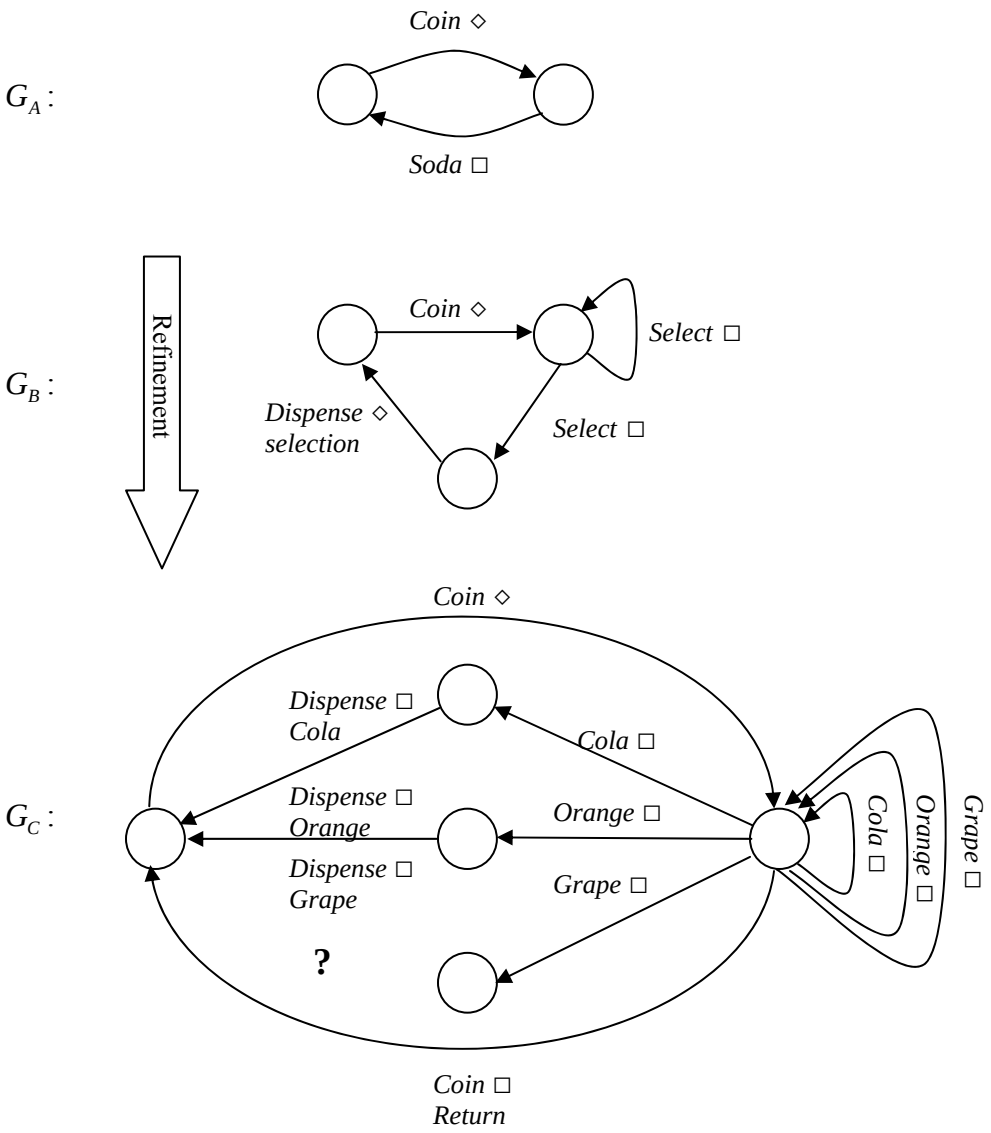



**Figure 11. DLTG Edge Refinements**

## E. SUMMARY

In this chapter we introduced Doubly Labeled Transition Graphs as well as Modal Constrained Bisimulation. Using a non-interference example in a Multi-Level System we

showed that Labeled Transition Graphs are inadequate when it comes to preserving security properties in a refinement. Having highlighted the need for Doubly Labeled Transition Graphs, we then went on to prove that Modal Constrained Bisimulation forms a Preorder on a set of DLTG refinements. Finally we discussed edge refinement in Doubly Labeled Transition Graphs, and proved that every edge of a graph that edge refines a “Must” edge also have to be “Must” edges.

# V. COMPOSITION OF DLTG REFINEMENTS

## A. INTRODUCTION

In this chapter we discuss composition of Labeled and Doubly Labeled Transition Graphs. In particular, we show when it is appropriate to compose graphs, and demonstrate how this can be carried out using a new join operation that we have constructed. We then demonstrate that it is always possible to join DLTGs refinements that stem from a common abstract graph. Subsequently we prove that the join of two DLTG refinements results in a graph that is a refinement of the abstract DLTG, and also serves as an abstract graph for the two DLTG refinements from which it was made. Finally, we prove that a set of DLTG refinements along with their abstract DLTG form an ideal.

## B. THE JOIN OF DLTG REFINEMENTS

Composition of Labeled and Doubly Transition Graphs is a very difficult subject. While much has been written concerning Labeled Transition Graphs and their usefulness in developing high assurance software, far less research has been carried out on how two compose these models. One of the principal difficulties when composing graphs is how one evaluates the states of the systems and then determines when states are equivalent. Bibighaus has suggested the use of Doubly Labeled Kripke Systems and Modal $\mu$ calculus [2, 19, 20, 21]. In Bibighaus' scheme each state of the system is defined by a set of predicates. Using ternary logic, it is then possible to reason about the properties of each state and then define a product operation between compatible states of two systems.

We suggest a different strategy. Our method of composition involves graph refinements that share bisimilar subgraphs as well as a common abstract graph. By utilizing the simulation relations between the refinements and the abstract graph, as well as the bisimulation relations between subgraphs of both refinements and the abstract graph, it is possible to speak of equivalent states without having to resort to the use of

state predicates. Moreover, because the refinements share a common abstract graph, we can verify that the resulting graph is well-formed and satisfies the desired properties of the abstract graph.

**Definition 5.1: (Compatible graph refinements)**

For all graphs $G_A, G_B$ and $G_C$ and $G_A', G_B', G_C'$ such that $G_B \lesssim_{R_1} G_A$, $G_C \lesssim_{R_2} G_A$, and such that their respective subgraphs satisfy $G_A' \approx_{R_1} G_B' \approx_{R_2} G_C'$ under the simulation relations $R_1$ and $R_2$, we say that $G_B$ and $G_C$ are join compatible.

**Definition 5.2: (The join of compatible graph refinements)**

Given three Labeled Transition Graphs $G_A, G_B$ and $G_C$ such that $G_B \lesssim_{R_1} G_A$, $G_C \lesssim_{R_2} G_A$, and $G_B$ and $G_C$ are join compatible. We create a new graph $G_B \vee G_C$ called the join of $G_B$ and $G_C$ as follows:

$$V_{G_B \vee G_C} = \{V_{G_A'} \cup V_{G_B}^{Join} \cup V_{G_C}^{Join}\}, \text{ where}$$

$$V_{G_B}^{Join} = \{V_{G_B} \setminus \{v \ni \exists v' \in V_{G_A'} \wedge (v, v') \in R_1\}\}$$

$$V_{G_C}^{Join} = \{V_{G_C} \setminus \{v \ni \exists v' \in V_{G_A'} \wedge (v, v') \in R_2\}\}$$

$$E_{G_B \vee G_C} = \{E_{G_B}^{Join} \cup E_{G_B}^{Connect} \cup E_{G_A'} \cup E_{G_C}^{Connect} \cup E_{G_C}^{Join}\}, \text{ where}$$

$$E_{G_B}^{Join} = \{e \in E_{G_B} : \eta(e) \wedge \tau(e) \in V_{G_B}^{Join}\}$$

$$E_{G_B}^{Connect} = \{e \in E_{G_B} : \eta(e) \vee \tau(e) \in V_{G_B}^{Join} \wedge \exists v' \in V_{G_A'} \ni (\eta(e), v') \in R_1 \vee (\tau(e), v') \in R_1\}$$

$E_{G_A'} = \{e \in E_{G_A'} : e \ni \eta(e) \wedge \tau(e) \in V_{G_A'}\}$, and $E_{G_C}^{Connect}$ and $E_{G_C}^{Join}$ are defined similarly to $E_{G_B}^{Connect}$ and $E_{G_B}^{Join}$.

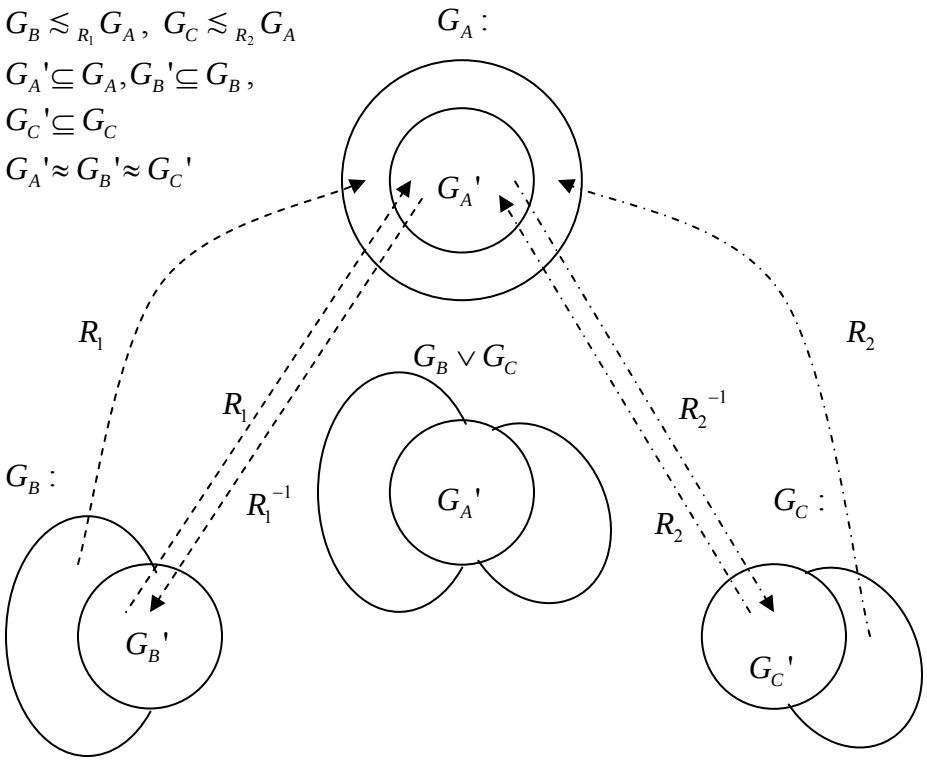



**Figure 12. Relationships Between Join Compatible Refinements**

**Theorem 5.1 (The join of two compatible graphs is commutative)**

$G_B \vee G_C = G_C \vee G_B$

**Proof:** Suppose there exist three Labeled Transition Graphs $G_A, G_B$ and $G_C$ such that $G_B \lesssim_{R_1} G_A$, $G_C \lesssim_{R_2} G_A$, and $G_B$ and $G_C$ are join compatible. Then $G_B \vee G_C = G_C \vee G_B$ follows directly from Definition 5.2.

**Lemma 5.1 A bisimulation subgraph for a set of refinements and a common abstract DLTG always exists.**

$$(G_A \not\approx_{R_1} G_B) \wedge (G_A \not\approx_{R_2} G_C) \Rightarrow \exists\, G_A' \subseteq G_A, G_B' \subseteq G_B, G_C' \subseteq G_C \ni (G_A' \approx_{R_1} G_B' \approx_{R_2} G_C')$$

**Proof:** Suppose there are three DLTGs such that $G_A \not\approx_{R_1} G_B$ and $G_A \not\approx_{R_2} G_C$.

From Definition 4.4 and Lemma 4.2 we have that: $\exists\, G_A^{[]}, G_B^{[]}|_{R_1^{-1}(E_{G_A}^{[]})}, G_C^{[]}|_{R_2^{-1}(E_{G_A}^{[]})} \ni G_A^{[]} \approx_{R_1} G_B^{[]}|_{R_1^{-1}(E_{G_A}^{[]})}$ and $G_A^{[]} \approx_{R_2} G_C^{[]}|_{R_2^{-1}(E_{G_A}^{[]})}$.

Moreover, from Theorem 2.1 we know that there exists a relation *R* such that: $G_A^{[]} \approx_{R_1} G_B^{[]}|_{R_1^{-1}(E_{G_A}^{[]})} \approx_R G_C^{[]}|_{R^{-1}(E_{G_A}^{[]})}$. Therefore we conclude that a bisimulation subgraph in the form of the abstract "Must" graph, and its associated images in the refinements will always exist for a set of refinements and a common abstract DLTG.

Lemma 5.1 is important because it tells us that we are always able to carry out a join of two DLTG refinements that share a common abstract graph. However, in order for the join operation to be us useful, we need to prove that the graph we obtain from the join operation will retain the properties of the abstract graph.

**Theorem 5.2 The join of two refinements of an abstract DLTG is also a refinement of the abstract DLTG.**

$$(G_A \not\approx_{R_1} G_B) \wedge (G_A \not\approx_{R_2} G_C) \Rightarrow \exists R \ni G_A \not\approx_R (G_B \vee G_C)$$

**Proof:** Suppose we have three DLTGs $G_A, G_B$ and $G_C$ such that $G_B \lesssim_{R_1} G_A$, $G_C \lesssim_{R_2} G_A$, and $G_A', G_B', G_C'$ are their respective subgraphs such that $G_A' \approx_{R_1} G_B' \approx_{R_2} G_C'$. Suppose further that we create a new DLTG $G_B \vee G_C$. Then in order to prove that the join of $G_B$ and $G_C$ also is a DLTG refinement of $G_A$ we must prove there exists a Left-Right relation *R* such that: $(G_B \vee G_C) \lesssim_R G_A$, and $G_A^{[]} \approx_R (G_B \vee G_C)^{[]}|_{R^{-1}(E_{G_A}^{[]})}$.

*Part* 1 *Defining* $R$:

We begin by defining the relation $R$. From Definition 5.2 we know that for every vertex $v \in V_{G_B \vee G_C}$, either $v \in V_{G_A'}$ or $v \in V_{G_B}^{Join}$ or $v \in V_{G_C}^{Join}$. If $v \in V_{G_A'}$, then clearly $v \in V_{G_A}$, therefore we use the identity mapping *I*. If $v \in V_{G_B}^{Join}$ then there exists a vertex $v' \in V_{G_A} \ni (v, v') \in R_1 \wedge (v', v) \in R_1^{-1}$, so we use $R_1$. Likewise, if $v \in V_{G_C}^{Join}$ then there exists a vertex $v' \in V_{G_A} \ni (v, v') \in R_2 \wedge (v', v) \in R_2^{-1}$, and we use $R_2$. We have therefore defined *R* and shown that it is Left-Right total.

*Part* 2 $(G_B \vee G_C) \lesssim_R G_A$:

Using the previously specified relation *R* we now show that for every edge *e* in $G_B \vee G_C$, there exists a corresponding edge $e'$ in $G_A$ under *R*. From Definition 4.1 we know that for every edge $e$ in $G_B \vee G_C$, $e \in E_{G_B \vee G_C}^{\diamond}$ or $e \in E_{G_B \vee G_C}^{[]}$. Moreover, from Definition 5.2 we have: $e \in E_{G_B}^{Join}$ or $e \in E_{G_B}^{Connect}$ or $e \in E_{G_A'}$ or $e \in E_{G_C}^{Connect}$ or $e \in E_{G_C}^{Join}$. Let $e$ be of the form $e = (v_1, \alpha, \mu, v_2)$, where $v_1, v_2 \in V_{G_B \vee G_C}$, $\alpha \in A_{G_B \vee G_C}$, and $\mu \in \{\Diamond, \Box\}$.

$e \in E_{G_B}^{Join}$:

If $e \in E_{G_B}^{Join}$, then By Definition 5.2 $v_1, v_2 \in V_{G_B}^{Join}$. Since we have $G_B \lesssim_{R_1} G_A$ and *R* is such that $\forall v \in V_{G_B}^{Join} : R = R_1$. We know that there exists a corresponding edge $e'$ in $G_A$.

$e \in E_{G_B}^{Connect}$:

If $e \in E_{G_B}^{Connect}$, then By Definition 5.2 $v_1 \in V_{G_B}^{Join} \wedge v_2 \in V_{G_A'}$ or $v_1 \in V_{G_A'} \wedge v_2 \in V_{G_B}^{Join}$. If $v_1 \in V_{G_B}^{Join} \wedge v_2 \in V_{G_A'}$ then using $R$ we know $\exists v_1' \in (v_1, v_1') \in R = R_1$,

$(v_2, v_2) \in I$ and there must exist a corresponding edge $e'$ in $G_A$. If $v_1 \in V_{G_A'} \wedge v_2 \in V_{G_B}^{Join}$, then similarly using $R$ we know that $(v_1, v_1) \in I$, $\exists v_1' \in (v_2, v_2') \in R = R_1$, and there must exist a corresponding edge $e'$ in $G_A$

$e \in E_{G_A'}$:

If $e \in E_{G_A'}$ then clearly $e \in E_{G_A}$ Using the identity mapping prescribed by $R$.

$e \in E_{G_C}^{Connect}$:

This argument is similar to that used for the case where $e \in E_{G_B}^{Connect}$. However, now we let $R = R_2$.

$e \in E_{G_C}^{Join}$:

This argument is similar to the one used for the case where $e \in E_{G_B}^{Join}$. However, in this case $R = R_2$.

*Part* 3 $G_A^{[]} \approx_R (G_B \vee G_C)^{[]}|_{R^{-1}(E_{G_A}^{[]})}$:

Using $R^{-1}$, we must demonstrated that $G_A^{[]} \approx_R (G_B \vee G_C)^{[]}|_{R^{-1}(E_{G_A}^{[]})}$. From Definition 5.2 and our initial assumptions, we know that $G_A'$ is a subgraph of $G_B \vee G_C$. Furthermore, from Lemma 5.1 we know that either $G_A^{[]}$ is a subgraph of $G_A'$ or $G_A' = G_A^{[]}$. The relation *R* dictates that $R = I$, which means that $R^{-1} = I$, since $I = I^{-1}$. If $G_A' = G_A^{[]}$, then $(G_B \vee G_C)^{[]}|_{R^{-1}(E_{G_A}^{[]})} = G_A^{[]}$, and under the identity mapping *I* it is clear that $G_A^{[]} \approx G_A^{[]}$. Let $R|_{G_A^{[]}}$ denote the restriction of the relation $R$ to the subgraph $G_A^{[]}$. Then, if $G_A^{[]}$ is a subgraph of $G_A'$ $R|_{G_A^{[]}} = I$, and we have $G_A^{[]} \lesssim_I G_A'$ and $G_A^{[]} \lesssim_{I^{-1}} G_A^{[]}$, which again means that $G_A^{[]} \approx_I G_A^{[]}$ and we have shown that $G_A^{[]} \approx_R (G_B \vee G_C)^{[]}|_{R^{-1}(E_{G_A}^{[]})}$.

Having proved all three sub-proofs using *R*, we have thus demonstrated that: $(G_A \mid\approx G_B) \wedge (G_A \mid\approx G_C) \Rightarrow G_A \mid\approx (G_B \vee G_C)$

We now demonstrate that the join of the two refinements also is an abstract graph for the two refinements from which it was created.

**Theorem 5.3 The join of two refinements of an abstract DLTG forms an abstract DLTG of the two refinements.**

$$(G_A \mid\approx_{R_1} G_B) \wedge (G_A \mid\approx_{R_2} G_C) \Rightarrow \exists R, R' \ni (G_B \vee G_C) \mid\approx_R G_B \wedge (G_B \vee G_C) \mid\approx_{R'} G_C$$

**Proof:** Suppose again that we have three DLTGs $G_A, G_B$ and $G_C$ such that $G_B \lesssim_{R_1} G_A$, $G_C \lesssim_{R_2} G_A$, and $G_A', G_B', G_C'$ are their respective subgraphs such that $G_A' \approx_{R_1} G_B' \approx_{R_2} G_C'$. Suppose further that we create a new DLTG $G_B \vee G_C$. In order to prove $(G_B \vee G_C) \mid\approx_R G_B$ we must show that there exist a Left-Right relation *R* such that: $G_B \lesssim_R (G_B \vee G_C)$ and $(G_B \vee G_C)^{[]} \approx_R G_B{}^{[]} \mid_{R^{-1}(E^{[]}_{G_B \vee G_C})}$.

.

*Part* 1 *Defining R*:

From Definition 5.2 we know that: $V_{G_B \vee G_C} = \{V_{G_A'} \cup V^{Join}_{G_B} \cup V^{Join}_{G_C}\}$, where

$V^{Join}_{G_B} = \{V_{G_B} \setminus \{v \ni \exists v' \in V_{G_A'} \wedge (v, v') \in R_1\}\}$

$V^{Join}_{G_C} = \{V_{G_C} \setminus \{v \ni \exists v' \in V_{G_A'} \wedge (v, v') \in R_2\}\}$

Furthermore, we know that:

$\forall v \in V_{G_B} : v \in V_{G_B'} \vee v \in \{V_{G_B} \setminus V_{G_B'}\}$

Mapping from $V_{G_B}$ to $V_{G_B \vee G_C}$ we have that:

If $v \in V_{G_B} \setminus V_{G_B'} \Rightarrow \exists v \in V^{Join}_{G_B} \ni (v, v) \in I$, so we use *I*.

Otherwise if $v \in V_{G_B'} \Rightarrow \exists v' \in V_{G_A'} \ni (v, v') \in R_1$, and we use $R_1$.

Mapping from $V_{G_B \vee G_C}$ to $V_{G_B}$ we have that:

If $v \in V_{G_B}^{Join} \Rightarrow \exists v \in \{V_{G_B} \setminus V_{G_B'}$ ə $(v,v) \in I$ , so we use $I$.

If $v \in V_{G_A'} \Rightarrow \exists v' \in V_{G_B'}$ ə $(v,v') \in R_1^{-1}$, so we use $R_1^{-1}$.

Finally, if $v \in V_{G_C}^{Join}$ then we must prove that there exists a corresponding vertex in $V_{G_B}$. This may initially seem impossible to do. However, since $G_B \lesssim_{R_1} G_A$ and $G_C \lesssim_{R_2} G_A$ it must be the case that $R_1$ and $R_2$ are both Left-Right total.

Therefore we have that $\forall v \in V_{G_C}^{Join} \exists v' \in V_{G_A}$ ə $(v,v') \in R_2$ and $\forall v' \in V_{G_A} \exists v'' \in V_{G_B}$ ə $(v',v'') \in R_1^{-1}$. Which in turn means that $\forall v \in V_{G_C}^{Join}$ $\exists v'' \in V_{G_B}$ ə $(v,v'') \in R_2 \circ R_1^{-1}$.

*Part 2* $G_B \lesssim_R (G_B \vee G_C)$:

Using the relation $R$ outlined above, we now show that $G_B \lesssim_R (G_B \vee G_C)$. From Definition 4.1 we know that for every edge $e$ in $G_B$, $e \in E_{G_B}^{\diamond} \vee e \in E_{G_B}^{[]}$. Moreover, $e \in E_{G_B'}$ or $e \in E_{G_B} \setminus E_{G_B'}$. Let $e$ be of the form $e = (v_1, \alpha, \mu, v_2)$, where $v_1, v_2 \in V_{G_B \vee G_C}$, $\alpha \in A_{G_B}$, and $\mu \in \{\Diamond, \Box\}$.

$e \in E_{G_B'}$:

From Definition 5.2 we know that $G_A' \subseteq G_B \vee G_C$. Furthermore, we know $G_A' \approx G_B'$, and $\forall v \in V_{G_B'} : R = R_1$. Therefore,

$e \in E_{G_B'} \Rightarrow (\exists v_1', v_2' \in V_{G_A'}$ ə $(v_1, v_1'), (v_2, v_2') \in R = R_1 \wedge$
$(e' = (v_1', \alpha, \mu, v_2')$ ə $e' \in E_{G_A}^{\diamond} \vee E_{G_A}^{[]}))$

$e \in E_{G_B} \setminus E_{G_B'}$:

If $e \in E_{G_B} \setminus E_{G_B'}$ then according to definition 5.2 and $R$, there are three possibilities:

1. $e \in E_{G_B} \setminus E_{G_B'} \Rightarrow \exists (v_1, v_1), (v_2, v_2) \in I \wedge e' = (v_1, \alpha, \mu, v_2) \ni e' \in E_{G_B}^{Join}$
2. $e \in E_{G_B} \setminus E_{G_B'} \Rightarrow \exists v_1 \in V_{G_B}^{Join} \wedge \exists v_2' \in V_{G_A'} \ni (v_1, v_1) \in I \wedge (v_2, v_2') \in R_1 \wedge e' = (v_1, \alpha, \mu, v_2') \in E_{G_B}^{Connect}$
3. $e \in E_{G_B} \setminus E_{G_B'} \Rightarrow \exists v_1' \in V_{G_A'} \wedge \exists v_2 \in V_{G_B}^{Join} \ni (v_1, v_1') \in R_1 \wedge (v_2, v_2) \in I \wedge e' = (v_1', \alpha, \mu, v_2) \in E_{G_B}^{Connect}$

Therefore we have shown that $G_B \lesssim_R (G_B \vee G_C)$.

*Part 3* $(G_B \vee G_C)^{[]} \approx_R G_B{}^{[]}|_{R^{-1}(E^{[]}_{G_B \vee G_C})}$:

Finally, we demonstrate that $R$ satisfies $(G_B \vee G_C)^{[]} \approx_R G_B{}^{[]}|_{R^{-1}(E^{[]}_{G_B \vee G_C})}$. From Definition 5.2, our initial assumptions, and part 1, we know that $G_A'$ is a subgraph of $G_B \vee G_C$, $G_B'$ is the subgraph of $G_B$, and $R$ ensures $G_A' \approx_{R_1} G_B'$. Furthermore, from Lemma 5.1 and Definition 4.3 we know that either $G_A' = G_A^{[]}$ or $G_A^{[]}$ is a subgraph of $G_A'$, and $G_B' = G_B^{[]}|_{R^{-1}(E^{[]}_{G_B \vee G_C})}$, or $G_B^{[]}|_{R^{-1}(E^{[]}_{G_B \vee G_C})}$ is a subgraph of $G_B'$. Since $G_A \not\approx_{R_1} G_B$ we know that $G_A^{[]} \not\approx_{R_1} G_B^{[]}|_{R_1^{-1}(E^{[]}_{G_A})}$. If $G_A' = G_A^{[]}$, then $(G_B \vee G_C)^{[]} \not\approx_R G_B^{[]}|_{R^{-1}(E^{[]}_{G_B \vee G_C})}$. Let $R^{-1}|_{G_A^{[]}}$ denote the restriction of the relation $R^{-1}$ to the subgraph $G_A^{[]}$. Then if $G_A^{[]}$ is a subgraph of $G_A'$, we use $R^{-1}|_{G_A^{[]}} = I$, where $I$ is the identity function and also have $(G_B \vee G_C)^{[]} \not\approx_R G_B^{[]}|_{R^{-1}(E^{[]}_{G_B \vee G_C})}$. We have therefore demonstrated that $(G_B \vee G_C)^{[]} \not\approx_R G_B^{[]}|_{R^{-1}(E^{[]}_{G_B \vee G_C})}$.

The proof that $(G_B \vee G_C) \not\approx_{R'} G_C$ is carried out similarly, and is therefore omitted. Having completed the last sub-proof we have proven that:

$(G_A \not\approx_{R_1} G_B) \wedge (G_A \not\approx_{R_2} G_C) \Rightarrow \exists R, R' \ni (G_B \vee G_C) \not\approx_R G_B \wedge (G_B \vee G_C) \not\approx_{R'} G_C$**).**

At this point, it would perhaps be prudent to give an example of the join procedure. To do so, we return to the refinement example used in Chapter 3. Recall that, after realizing that our initial model of a flashlight was incomplete, we created several new graphs. One of them took into account the battery running out of electrical charge, while another considered bulb failure. Finally, we created a graph that incorporated both of these types of failures. Although it wasn't mentioned at the time, what we actually did was to join two of the graphs and create the graph that included both failures. We were able to do this because the two refinements and the abstract graph shared the same subgraph which included the $v_{Off}$ and $v_{On}$ along with two *Push* edges. To see how this was carried out consider Figure 13.

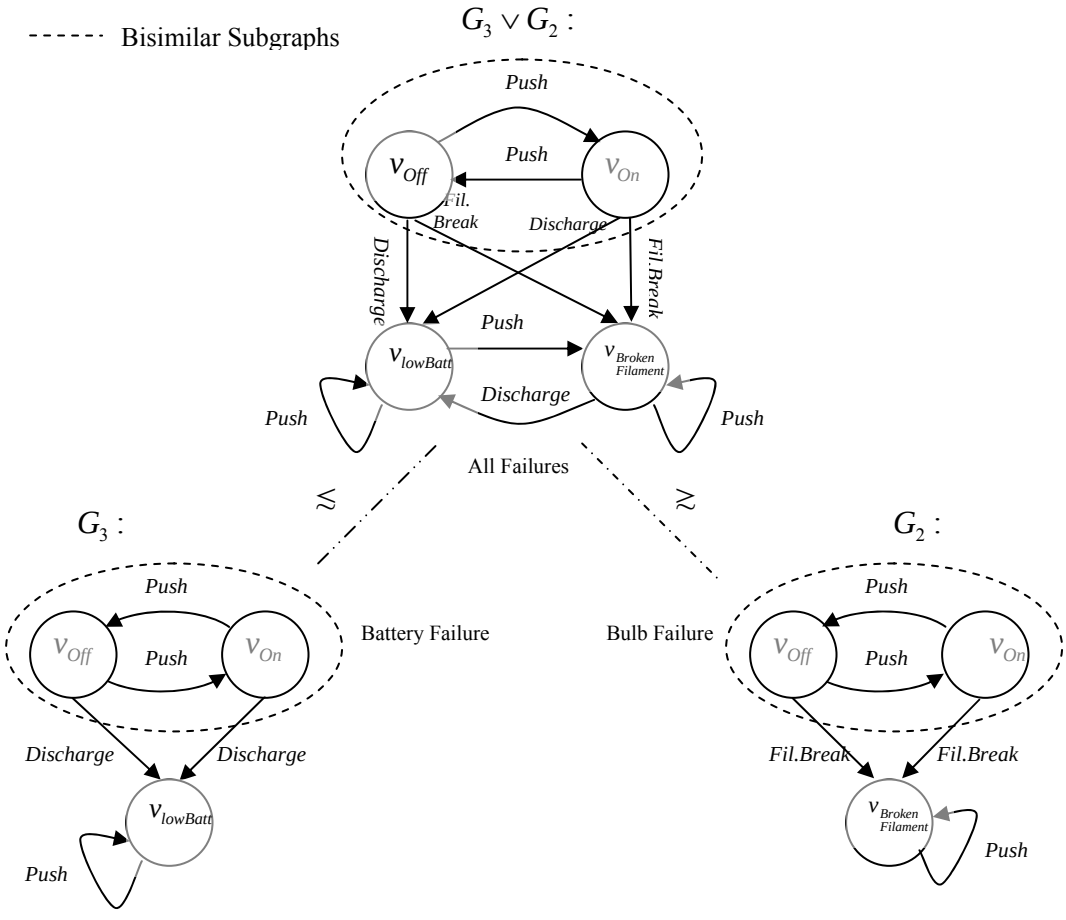



**Figure 13. The Join Of Two LTGs**

**Definition 5.3: (An ideal [22])**

A non-empty subset $I$ of a partially ordered set is said to be an ideal, if the following conditions hold:

1. $(a \in I) \wedge (b \in I) \Rightarrow (a \vee b) \in I)$.

2. $(a \leq b) \wedge (b \in I) \Rightarrow (a \in I)$

Moreover, $(a \vee b) \in I \Rightarrow (a \in I) \wedge (b \in I)$ and $a \leq (a \vee b)$ and $b \leq (a \vee b)$

**Theorem 5.4 A set of refinements of a common abstract DLTG forms an ideal.**

**Proof:** Let $I$ represent the set of DLTG refinements of a DLTG $G_A$. From Theorem 4.1 we know that MCB Forms a Preorder on a set of DLTG Refinements. The rest of the proof follows directly by using Lemma 5.1, Theorems 4.1, 5.2, 5.3, and applying Definition 5.3.

## C. SUMMARY

In this chapter we discussed composition of Labeled and Doubly Labeled Transition Graphs. We showed that two refinements that share a common abstract graph as well as a subgraph, can be joined together to form a new graph. We further demonstrated that it is always possible to join DLTG refinements, provided they share a common abstract graph. Subsequently we proved the graph obtained by joining two refinements, not only is a refinement of the abstract graph, but that it also represents an abstract graph for the refinements from which it was made. Finally, we proved that the set of refinements and their abstract graph form an ideal.

THIS PAGE INTENTIONALLY LEFT BLANK

# VI. FUTURE WORK

In this thesis, we have developed a method of composing refinements that share a common abstract graph. One fundamental question that still remains is how to go about composing graphs that do not share a common abstract graph. We believe that this may also be possible to accomplish using the join method we developed. In order to do so, we suggest first attempting to join the two abstract graphs. This is presumably possible, since composition of graphs only seems meaningful if the graphs share some common structure; albeit perhaps very little. Once a new abstract graph has been achieved, we hypothesize that the joining of the refinements will be possible. Should this prove to be the case, it could conceivably also be possible to address the dual concept of decomposition using our notion of graph joins. In this case one would split an abstract graph into several abstract subgraphs each containing common structure. Having done so, it would therefore be possible to join their respective refinements back together at a later stage of the refinement process.

THIS PAGE INTENTIONALLY LEFT BLANK

# VII. CONCLUSION

In this thesis we have explained the work of Bibighaus using a graph theoretic framework. We began by defining a Labeled Transition Graph along with the notion of traces and trace equivalence. We introduced simulation and bisimulation morphisms between graphs, and proved that bisimulation not only guarantees trace equivalence, but that it also forms an equivalence relation on a set of Labeled Transition Graphs. We then discussed the notion of abstraction and refinement, and explained why refinement is important when developing high assurance software. By first developing an abstract model and then refining it, we hope to achieve a concrete implementation that upholds the properties set forth in the abstract specification. The proof that this is in fact the case, takes the form of a simulation mapping between the refinement and the abstract graph. Using an example involving non-interference we then demonstrated that the Labeled Transition Graph model is inadequate when it comes to ensuring that certain security properties are retained throughout the refinement process. This in turn motivated the introduction of Doubly Labeled Transition Graphs. By specifying which edges must occur in a refinement as opposed to those that may or may not occur, we were able to define a new mapping scheme. This type of morphism, which we refer to as Modal Constrained Bisumulation (MCB), guarantees not only that all edges of the refinement graph map to edges in the abstract graph, but additionally that a subset of the edges of the abstract graph always map to edges in the refinement. Accordingly, Doubly Labeled Transition Graphs coupled with the use of the Modal Constrained Bisimulation mapping, ensures that we are able to retain a larger set of properties throughout the refinement process, thereby increasing the chances that our concrete implementation will conform to specifications.

In addition to explaining Bibighaus' work we contributed to the subject matter with the following results: First we proved that in order to guarantee that a sequence of refinements retain the desired properties of the abstract graph it is necessary that every edge of a graph that edge refines a "Must" edge of a DLTG, also is a must edge. Secondly, and perhaps most importantly, we developed a form of graph composition

called the "join" of two graphs, whereby two refinements that share a common subgraph along with the same abstract graph are joined together to produce a new graph. As a result of the Modal Constrained Bisimulation involved in the DLTG refinement process, we demonstrated that it is always possible to carry out this type of composition, provided the refinements share a common abstract graph. Furthermore, we went on to prove that the resulting from the join operation is not only a refinement of the abstract graph, but that it also represents an abstract graph for the refinements from which it was created.

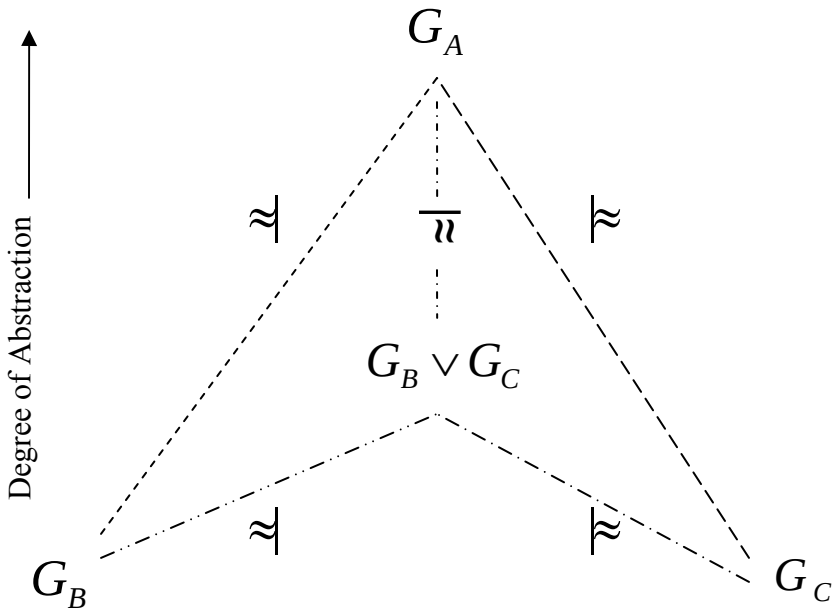



**Figure 14. DLTG Refinements and Their Abstract DLT Form an Ideal**

Finally we showed that a set of DLTG refinements along with their abstract graph form and ideal.

These results are relevant because they tell us that, if during the course of the development process, we end up with refinements that individually capture desirable aspects of a system, it is possible to combine them into model that incorporates all of the features of both refinements, while still satisfying the properties of the abstract model. Inasmuch, we hope that this work will prove useful for future high assurance software development processes that utilize the Doubly Labeled Transition System framework.

# LIST OF REFERENCES


[1] Common Criteria Project Sponsoring Organizations. *Common Criteria for Information Technology Security Evaluation Part1: Introduction and general model*, version 2.2 revision 256 edition, January 2004.

[2] Dave L. Bibighaus. *Applying Doubly Labeled Transition Systems to the Refinement Paradox.* Ph.D. dissertation, Naval Postgraduate School, 2005.

[3] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, London, 1985.

[4] Robin Milner. *Communication and Concurrency.* Prentice Hall, Hertfordshire, 1989.

[5] David A. Schmidt. From trace sets to modal-transition systems by setwise abstract interpretation. In Workshop on *Structure-Preserving Relations*, Amagasaaki, Japan, March 2001. Elsevier Electronic Notes in Theoretical Computer Science.

[6] Sam Owre, N Shankar, J.M. Rushby and D. W. J. Stringer-calvert. *PVS Language Reference Version 2.4.* SRI International , 333 Ravenswood Ave, Menlo Park, CA 94025, 2001.

[7] G. Chartrand and P. Zhang, *Introduction to Graph Theory*, McGraw Hill, Boston, 2005, p. 11.

[8] Faron Moller and Scott A. Smolka. On the computational complexity of bisimulation. *ACM Comput. Surv*., 27(2), pp. 287-289, June 1995.

[9] R. Paige and R.E. Tarjan. Three partition refinement algorithms.

*SIAM Journal of Computing*, 16(6), pp. 973-989, December 1987.

[10] Linda Null and Julia Lobur. *The Essentials of Computer Organization and Architecture*, Jones and Bartlett, Boston, 2003, p. 26.

[11] Anthony Hall and Roderick Chapman. Correctness by Construction: Developing a Commercial Secure System. IEEE Software, pp 18-25, Jan/Feb 2002.

[12] John Derrick and Eerke Boiten. Recent advances in refinement. In Egon Borger, Angelo Gargantini, and Elvini Riccobene, editors, *Abstract State Machines 2003*, March 2003, pp. 33-56.

[13] John Derrick, Eerke Boiten , Howard Bowman, and Maarten Steen. Week refinement in Z. In J.P. Bowen, M.G. Hinchey, and D. Till, editors, *ZUM'97: The Z Formal Specification Notation, volume 1212 of Lecture Notes in Computer Science*, pp. 369-388, April 1997.

[14] Annalisa Bossi, Carla Piazza, and Sabina Rossi. Preserving (security) properties under action refinement. In *Convegno Italiano di Logica Computazionale*, Universita di Parma, 2004.

[15] Kim G. Larsen and Bent Thomsen. A model process logic. *Third Annual Symposium on Logic in Computer Science*, Edinburgh, 1988.

[16] Dennis Dams. *Abstract Interpretation and Partial Refinement For Model Checking*. PhD dissertation, Technische Universiteit Eindhoven, The Netherlands, 1996.

[17] George W. Dinolt. "Security Defined by Doubly Labeled Transition Systems." Class Lecture Slides. CS 4605 Department of Computer Science, Naval Postgraduate School, Monterey CA, Summer 2007.

[18] Joseph Goguen and J. Mesenguer. Security policies and Security models. *IEEE Symposium on Security and Privacy*, pp. 11-20, April 1982.

[19] David A. Schmidt. From trace sets to modal-transition systems by setwise abstract interpretation. *Workshop on Structure-Preserving Relations*, Amagasaaki, Japan, March 2001. Elsevier Electronic Notes in Theoretical Computer Science.

[20] Michael Huth, Radha Jagadeesan, and David A. Schmidt. Modal transition systems: A foundation for three –valued program analysis. *Proc. European Symposium on Programming*, pp. 155-169, 2001.

[21] David A. Schmidt. Structure-preserving binary relations for program abstraction. In T. Morgensen, d. Schmidt, and I.H. Sudburough, editors, *The Essence of Computation: Complexity, Analysis, Transformation*. Springer LNCS 2566, 2002.

[22] K. Kuratowski and A. Mostowski. *Set Theory With an Introduction To Descriptive Set Theory.* PWN Polish Scientific Publishers, Warsaw, 1976, pp. 158-159.

THIS PAGE INTENTIONALLY LEFT BLANK

# INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dr. George W. Dinolt
   Naval Postgraduate School
   Monterey, California

4. Dr. Harold M. Fredricksen
   Naval Postgraduate School
   Monterey, California

5. Dr. Relucca Gera
   Naval Postgraduate School
   Monterey, California

6. Mr. Edward V. Ziegler
   National Security Agency
   Fort George G. Meade, Maryland

7. Dr. David L. Bibighaus
   RADC, Griffith AFB
   Rome, New York